Equational Formulas and Pattern Operations in Initial Order-Sorted Algebras

José Meseguer and Stephen Skeirik

Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract. A *pattern* t, i.e., a term possibly with variables, denotes the set (language) $[\![t]\!]$ of all its *ground instances*. In an untyped setting, symbolic operations on finite sets of patterns can represent Boolean operations on languages. But for the more expressive patterns needed in declarative languages supporting rich type disciplines such as subtype polymorphism untyped pattern operations and algorithms break down. We show how they can be properly defined by means of a signature transformation $\Sigma \mapsto \Sigma^{\#}$ that enriches the types of Σ. We also show that this transformation allows a systematic reduction of the first-order logic properties of an initial order-sorted algebra supporting subtype-polymorphic functions to equivalent properties of an initial many-sorted (i.e., simply typed) algebra. This yields a new, simple proof of the known decidability of the first-order theory of an initial order-sorted algebra.
Keywords: pattern operations, initial decidability, order-sorted logic.

1 Introduction

Term patterns are used everywhere in functional and logic programming: to define predicates and functions, to perform automated deduction tasks like rewriting, matching, unification, resolution, and Knuth-Bendix completion, and also as a *symbolic notation* to describe *languages* as sets of term instances, and *language operations* by corresponding symbolic operations on the term patterns defining them. Such pattern operations, first systematically studied by Lassez and Marriott in [13] and further studied in, e.g., [12,19,18,7] have many applications to, e.g., machine learning, negation in logic programming, sufficient completeness of function definitions, inductive theorem proving, and automated model building.

For greater expressiveness many declarative languages support rich type disciplines. This holds true for both higher-order functional languages and rule-based languages. For example, OBJ [11], CafeOBJ [8], and Maude [2] all support types, subtypes, subtype polymorphism, and —through their parameterized types— polymorphic and dependent types. Obviously, all the above-mentioned applications of pattern operations are also needed for these languages. What is not at all obvious —and to the best of our knowledge does not seem to have been investigated so far— is whether the algorithms defining the Boolean algebra of pattern operations for the *untyped* case in, e.g., [13,12,19,18,7] extend in a straightforward way to the more expressive patterns now available in these richer type disciplines. The example below clearly shows that they do not.

The graph on the left describes an *order-sorted signature* [10] with two types, A and B, and a subtype inclusion $A < B$ depicted by the vertical bar. f is *subtype polymorphic*, with two typings: $f : A \to A$, and $f : B \to B$. We have constants a, b of respective types A, B. A *pattern* t, i.e., a term possibly with variables, denotes the set (language) $[\![t]\!] = \{t\sigma \mid \sigma \ \ground\}$ of all its *ground instances*. The symbolic pattern difference $t - t'$ denotes the language $[\![t - t']\!] = [\![t]\!] - [\![t']\!]$. In the untyped case, it is well-known [13] that when t and t' are *linear patterns* (have no repeated variables), the symbolic difference $t - t'$ always denotes a language expressible as $[\![u_1]\!] \cup \ldots \cup [\![u_k]\!]$, for $\{u_1, \ldots, u_k\}$ a finite set of patterns. If this were to hold in the order-sorted case, it should hold, in particular, for $[\![x{:}B - y{:}A]\!]$, with $x{:}B, y{:}A$ variables of sorts A, B. Adopting the convention $f^0(x) = x$, we have, $[\![y{:}A]\!] = \{f^n(a) \mid n \geq 0\}$, and $[\![x{:}B]\!] = [\![y{:}A]\!] \cup \{f^n(b) \mid n \geq 0\}$. Therefore, $[\![x{:}B - y{:}A]\!] = \{f^n(b) \mid n \geq 0\}$. But *there is no finite set of patterns* $\{u_1, \ldots, u_k\}$ such that $[\![u_1]\!] \cup \ldots \cup [\![u_k]\!] = \{f^n(b) \mid n \geq 0\}$. Indeed, the only possible choice for a u_i is $u_i = b$. All other choices: $u_i = a$, $u_i = x'{:}B$, $u_i = y'{:}A$, $u_i = f^{n+1}(x'{:}B)$, or $u_i = f^{n+1}(y'{:}A)$, $n \geq 0$, are impossible.

Is all lost? Not if we make our signature more expressive: the graph on the right adds a new subtype $B^\# < B$, lowers the typing of b to $B^\#$, and adds the typing $f : B^\# \to B^\#$. Now $[\![z : B^\#]\!] = \{f^n(b) \mid n \geq 0\}$. and we can *symbolically compute* the difference $x : B - y : A = z : B^\#$. This example shows that the problem is insoluble *as formulated*, but it can be solved by a *signature transformation* extending the original signature Σ. In Section 3 we formally define such a transformation $\Sigma \mapsto \Sigma^\#$ that enriches a finite order-sorted signature Σ with additional sorts like the sort $B^\#$ above. This is a key step for obtaining a Boolean algebra of *order-sorted* patterns in Section 5.

But the $\Sigma \mapsto \Sigma^\#$ transformation has other far-reaching consequences. Since it is well-known that pattern operations are intimately connected with first-order logic formulas and with negation elimination in such formulas [12,19,18,7], we should first of all ask what light can the $\Sigma \mapsto \Sigma^\#$ transformation shed on the *validity of formulas in initial order-sorted algebras*. As we show in Section 4, it sheds a lot of light: it makes the validity of a first-order formula in an initial *order-sorted* algebra equivalent to the validity of an associated formula in an associated *many-sorted* initial algebra. Since the first-order theory of a many-sorted initial algebra is well-known to be decidable [14,15,3], this proves the decidability of the first-order theory of an initial order-sorted algebra. This result goes back to [4,5], but the proof obtained through the $\Sigma \mapsto \Sigma^\#$ transformation is considerably simpler. Furthermore, it provides a new, general *transfer principle* to *reduce* certain order-sorted algebra problems to many-sorted algebra ones.

We put this transfer principle to work for order-sorted pattern operations in Section 5, where we show that they can be *reduced* to operations on *many-sorted* $\Sigma^{\#}$-patterns. Furthermore, we develop an intrinsically *order-sorted* algorithm for pattern operations based on the signature $\Sigma \cup \Sigma^{\#}$ that enjoys important advantages. As reported in Section 6, we have implemented this algebra of order-sorted pattern operations in Maude using reflection. Proofs and some additional details are included in appendices.

Acknowledgements. Partially supported by NSF Grant CNS 13-19109. We thank the referees for their excellent suggestions to improve the paper.

2 Preliminaries on Order-Sorted Algebra

The following material is adapted from [16], which generalizes [10]. It summarizes the basic notions of order-sorted algebra needed in the rest of the paper. It assumes the notions of many-sorted signature and many-sorted algebra, e.g., [6].

Definition 1. *An* order-sorted signature *is a triple $\Sigma = (S, \leq, \Sigma)$ with $(S, \leq)$ a poset and (S, Σ) a many-sorted signature.*

$\hat{S} = S/\equiv_{\leq}$, called the set of connected components *of $(S, \leq)$, is the quotient of S under the smallest equivalence relation $\equiv_{\leq} = (\leq \cup \geq)^{+}$. The order $\leq$ and equivalence $\equiv_{\leq}$ are extended to sequences of the same length in the usual way, e.g., $s'_1 \ldots s'_n \leq s_1 \ldots s_n$ iff $s'_i \leq s_i$, $1 \leq i \leq n$.*

Σ is called sensible *(resp.* monotonic*) if for any two operators $f : w \to s, f : w' \to s' \in \Sigma$, with w and w' of same length, we have $w \equiv_{\leq} w' \;\Rightarrow\; s \equiv_{\leq} s'$. (resp. $w \geq w' \;\Rightarrow\; s \geq s'$). Note that a* many-sorted signature *Σ is the special case in which the poset $(S, \leq)$ is discrete, i.e., $s \leq s'$ iff $s = s'$.*

For connected components $[s_1], \ldots, [s_n], [s] \in \hat{S}$

$$f^{[s_1]\ldots[s_n]}_{[s]} = \{f : s'_1 \ldots s'_n \to s' \in \Sigma \mid s'_i \in [s_i] \;\; 1 \leq i \leq n, \;\; s' \in [s]\}$$

is the family of "subsort polymorphic" operators f for those components. □

We will assume throughout that *each connected component $[s] \in \hat{S}$ contains a top element $\top_{[s]} \in [s]$* such that for each $s' \in [s]$, $\top_{[s]} \geq s'$.

Definition 2. *For $\Sigma = (S, \leq, \Sigma)$ an OS signature, an* order-sorted Σ-algebra *A is a many-sorted (S, Σ)-algebra A such that:*

- *whenever $s \leq s'$, then we have $A_s \subseteq A_{s'}$, and*
- *whenever $f : w \to s, f : w' \to s' \in f^{[s_1]\ldots[s_n]}_{[s]}$ and $\overline{a} \in A^w \cap A^{w'}$, then we have $A_{f:w\to s}(\overline{a}) = A_{f:w'\to s'}(\overline{a})$, where $A^{s_1\ldots s_n} = A_{s_1} \times \ldots \times A_{s_n}$.*

An order-sorted Σ-homomorphism *$h : A \to B$ is a many-sorted (S, Σ)-homomorphism such that whenever $[s] = [s']$ and $a \in A_s \cap A_{s'}$, then we have $h_s(a) = h_{s'}(a)$. h is* injective*, resp.* surjective*, resp.* bijective*, iff for each $s \in S$ h_s is injective, resp. surjective, resp. bijective. We call h an* isomorphism *if there is another order-sorted Σ-homomorphism $g : B \to A$ such that for each $s \in S$, $h_s \circ g_s = 1_{B_s}$,*

and $g_s \circ h_s = 1_{A_s}$, with $1_{A_s}, 1_{B_s}$ the identity functions on A_s, B_s. If each $[s] \in \hat{S}$ has a top element $\top_{[s]}$, one can show that f is an isomorphism iff f is bijective. Σ-algebras and homomorphisms define a category $\mathbf{OSAlg}_\Sigma$. □

Theorem 1. *[16] The category* $\mathbf{OSAlg}_\Sigma$ *has an initial algebra. Furthermore, if Σ is sensible, then the term algebra T_Σ with:*

- *if $a : \epsilon \to s$ then $a \in T_{\Sigma,s}$, (ϵ denotes the empty string),*
- *if $t \in T_{\Sigma,s}$ and $s \leq s'$ then $t \in T_{\Sigma,s'}$,*
- *if $f : s_1 \dots s_n \to s$ and $t_i \in T_{\Sigma,s_i}$ $1 \leq i \leq n$, then $f(t_1, \dots, t_n) \in T_{\Sigma,s}$,*

is initial, i.e., there is a unique Σ-homomorphism to each Σ-algebra.

Say Σ *has non-empty sorts* iff for each $s \in S$, $T_{\Sigma,s} \neq \emptyset$. To ensure $[\![t]\!] \neq \emptyset$ for any term t *we will assume throughout that* Σ has non-empty sorts.

An S-sorted set $X = \{X_s\}_{s \in S}$ of *variables*, satisfies $s \neq s' \Rightarrow X_s \cap X_{s'} = \emptyset$, and the variables in X are always assumed disjoint from all constants in Σ. The Σ-*term algebra* with variables in X, $T_\Sigma(X)$, is the *initial algebra* for the signature $\Sigma(X)$ obtained by adding to Σ the variables in X *as extra constants*. Since a $\Sigma(X)$-algebra is just a pair (A, α), with A a Σ-algebra, and α an *interpretation of the constants* in X, i.e., an S-sorted function $\alpha \in [X \to A]$, the $\Sigma(X)$-initiality of $T_\Sigma(X)$ can be expressed as the following corollary of Theorem 1:

Theorem 2. *(Freeness Theorem). If Σ is sensible, for each $A \in \mathbf{OSAlg}_\Sigma$ and $\alpha \in [X \to A]$, there exists a unique Σ-homomorphim, $\_\alpha : T_\Sigma(X) \longrightarrow A$ extending α, i.e., such that for each $s \in S$ and $x \in X_s$ we have $x\alpha_s = \alpha_s(x)$.*

The first-order language of *equational Σ-formulas* is defined in the usual way: its atoms are Σ-*equations* $t = t'$, where $t, t' \in T_\Sigma(X)_{\top_{[s]}}$ for some $[s] \in \hat{S}$ and each X_s is assumed countably infinite. The set $Form(\Sigma)$ of *equational Σ-formulas* is then inductively built from atoms by: conjunction ($\wedge$), disjunction ($\vee$) negation ($\neg$), and universal ($\forall x{:}s$) and existential ($\exists x{:}s$) quantification with sorted variables $x{:}s \in X_s$ for some $s \in S$. The literal $\neg(t = t')$ is denoted $t \neq t'$.

The *satisfaction* relation between Σ-algebras and formulas is defined in the usual way: given a Σ-algebra A, a formula $\varphi \in Form(\Sigma)$, and an assignment $\alpha \in [Y \to A]$, with $Y = fvars(\varphi)$ the free variables of φ, we define the satisfaction relation $A, \alpha \models \varphi$ inductively as usual: for atoms, $A, \alpha \models t = t'$ iff $t\alpha = t'\alpha$; for Boolean connectives it is the corresponding Boolean combination of the satisfaction relations for subformulas; and for quantifiers: $A, \alpha \models (\forall x{:}s)\ \varphi$ (resp. $A, \alpha \models (\exists x{:}s)\ \varphi$) holds iff for all $a \in A_s$ (resp. there is an $a \in A_s$) we have $A, \alpha \uplus \{(x{:}s, a)\} \models \varphi$, where the assignment $\alpha \uplus \{(x{:}s, a)\}$ extends α by mapping $x{:}s$ to a. Finally, $A \models \varphi$ holds iff $A, \alpha \models \varphi$ holds for each $\alpha \in [Y \to A]$, where $Y = fvars(\varphi)$. We say that φ is *valid* (or *true*) in A iff $A \models \varphi$.

Definition 3. *A* signature morphism $H : \Sigma \to \Sigma'$ *(called a* view *in Maude [2]) is a monotonic function $H : (S, \leq) \to (S', \leq')$ of the underlying posets of sorts, together with a mapping H sending each $f : s_1 \dots s_n \to s$ in Σ to a term $H(f) \in T_{\Sigma'}(\{x_1 : H(s_1), \dots, x_n : H(s_n)\})_{H(s)}$. H defines a well-typed*

translation of the syntax of Σ into that of Σ'. It inductively maps each Σ-term t to a Σ'-term $H(t)$ by mapping $x:s$ to $x:H(s)$, and $H(f(t_1,\ldots,t_n))$ to $H(f)\{x_1:H(s_1) \mapsto H(t_1),\ldots,x_n:H(s_n) \mapsto H(t_n)\}$, where $\{x_1:H(s_1) \mapsto H(t_1),\ldots,x_n:H(s_n) \mapsto H(t_n)\}$ denotes the obvious substitution. H extends naturally to a translation of equational formulas $H : Form(\Sigma) \to Form(\Sigma')$ by mapping atoms according to H, respecting Boolean connectives, and mapping each quantifier $\forall x{:}s$ (resp. $\exists x{:}s$) to $\forall x{:}H(s)$ (resp. $\exists x{:}H(s)$).

A signature inclusion*, denoted $\Sigma \hookrightarrow \Sigma'$, is a signature morphism that is a poset inclusion $(S,\leq) \hookrightarrow (S',\leq')$ on sorts and maps each $f : s_1 \ldots s_n \to s$ to itself: more precisely, to the term $f(x_1{:}s_1,\ldots,x_n{:}s_n)$.* □

A signature morphism $H : \Sigma \to \Sigma'$ induces a functor in the *opposite* direction $\_|_H$: $\mathbf{OSAlg}_{\Sigma'} \to \mathbf{OSAlg}_{\Sigma}$, where for each $B \in \mathbf{OSAlg}_{\Sigma'}$, the algebra $B|_H \in \mathbf{OSAlg}_{\Sigma}$, called its *$H$-reduct*, is defined using H as follows: (i) for each $s \in S$, $(B|_H)_s = B_{H(s)}$; and (ii) for each $f : s_1 \ldots s_n \to s$ in Σ, $(B|_H)_f$ is the function $\lambda(x_1 \in B_{H(s_1)},\ldots,x_n \in B_{H(s_n)})$. $H(f) : B_{H(s_1)} \times \ldots \times B_{H(s_n)} \to B_{H(s)}$ defined by the term $H(f)$ in the Σ'-algebra B.

In Goguen and Burstall's sense, the key point about order-sorted signature morphisms is that they make order-sorted logic an *institution* [9], so that *truth is preserved along translations*. That is, for any $B \in \mathbf{OSAlg}_{\Sigma'}$ and any sentence (i.e., $fvars(\varphi) = \emptyset$) $\varphi \in Form(\Sigma)$ we have the equivalence:

$$(\dagger) \qquad B \models H(\varphi) \;\Leftrightarrow\; B|_H \models \varphi.$$

This equivalence can be checked in several ways. For example, one can use the embedding of order-sorted logic in membership equational logic, itself embedded in many-sorted first-order logic, as detailed in [16]. This reduces the issue to the same well-known equivalence for many-sorted first-order logic.

An important requirement on a sensible and monotonic signature is *regularity* [10]. It requires for each $f \in \Sigma$ and $u \in S^*$ that, if the set $\{ws \in S^* \mid f : w \to s \in \Sigma \wedge w \geq u\}$ is non-empty, then it has a smallest element. Regularity (or just *preregularity* [2]) ensures that each Σ-term $t \in T_\Sigma(X)$ has a *least sort*, denoted $ls_\Sigma(t)$, with $t \in T_\Sigma(X)_{ls_\Sigma(t)}$. This makes order-sorted automated deduction tasks like term rewriting or unification much easier: the matching of a term t to a variable $x{:}s$ will succeed iff $ls_\Sigma(t) \leq s$. Without regularity, or preregularity, a costly determination of all possible sorts of t is needed.

3 The $\Sigma \mapsto \Sigma^\#$ Signature Transformation

We define a signature transformation $\Sigma \mapsto \Sigma^\#$ that will give us the key to study validity of equational formulas in initial order-sorted algebras in Section 4 and pattern operations in Section 5. Σ is a regular order-sorted finite signature with poset of sorts $(S,\leq)$. As first remarked by H. Comon-Lundh in [4], an order-sorted signature Σ is just a Σ^u-tree automaton, with Σ^u the unsorted version of Σ, set of states S, and transitions rules: (i) $f(s_1,\ldots,s_n) \to s$ for each $f : s_1 \ldots s_n \to s$ in Σ, and (ii) ϵ-rules $s \to s'$ for each $s < s'$ in $(S,\leq)$. $T_{\Sigma,s}$ is

the language accepted by the accepting state s. This means that the problem of whether $T_{\Sigma,s} = \emptyset$, or whether any Boolean combination of sets $T_{\Sigma,s_1}, \ldots, T_{\Sigma,s_n}$ is empty, are problems decidable by an emptiness check on a regular tree language.

To construct $\Sigma^\#$ we must first define its set $S^\#$ of sorts. Call $s \in S$ *atomic* iff s is a minimal element in the poset $(S, \leq)$. The key idea is to add to S new atomic sorts $s^\#$ characterizing all terms whose least sort is exactly s, where s is non-atomic. But we want $s^\#$ to be non-empty. Let $\downarrow s = \{s' \in S \mid s' < s\}$, and $glbs(s)$ the maximal elements of $\downarrow s$. Call $s \in S$ *redundant* iff $T_{\Sigma,s} - \bigcup_{s' \in glbs(s)} T_{\Sigma,s'} = \emptyset$. We only add $s^\#$ to $S^\#$ if s is non-atomic and irredundant. Since non-emptiness is decidable, we can effectively construct $S^\#$ as the set containing all atomic sorts in S and all new sorts $s^\#$ with $s \in S$ non-atomic and irredundant.

We want a *many-sorted* signature $\Sigma^\#$ on sorts $S^\#$ such that: (i) for s an atomic sort in Σ, we have $T_{\Sigma^\#,s} = T_{\Sigma,s}$, (ii) for each $s^\# \in S^\#$ we have $T_{\Sigma^\#,s^\#} = T_{\Sigma,s} - \bigcup_{s' \in glbs(s)} T_{\Sigma,s'}$; and (iii) if $s, s' \in S^\#$ and $s \neq s'$, then $T_{\Sigma^\#,s} \cap T_{\Sigma^\#,s'} = \emptyset$. Thus, we will be able to represent each sort $s \in S$ as a *disjoint union* of sorts in $S^\#$. That is, define the function $atoms : S \to \mathcal{P}(S^\#)$ inductively as follows: $atoms(s) =$ **if** s is atomic **then** $\{s\}$ **else if** s is irredundant **then** $\{s^\#\} \cup atoms(s_1) \cup \ldots atoms(s_n)$ **else** $atoms(s_1) \cup \ldots atoms(s_n)$ **fi fi**, where $glbs(s) = \{s_1, \ldots, s_n\}$. It then follows from (i)–(iii) above that for any $s \in S$ we will have:

$$T_{\Sigma,s} = \biguplus_{s' \in atoms(s)} T_{\Sigma^\#,s'}$$

This is what we want. We still have to define $\Sigma^\#$. For this, it is useful to decompose Σ as a "telescope" $\Sigma_0 \subset \Sigma_1 \subset \ldots \Sigma_{k-1} \subset \Sigma$. We assume that each constant $a : \epsilon \to s$ in Σ has a single declaration of the specified sort s. To simplify the $\Sigma^\#$ construction we also assume, without real loss of generality, that Σ can have "subsort overloading" but does not have any "ad-hoc overloading;" that is, if $(f : s_1 \ldots s_n \to s), (f : s'_1 \ldots s'_n \to s') \in \Sigma$ then $[s_i] = [s'_i]$ $1 \leq i \leq n$, and $[s] = [s']$. Recall the notation $f^{[s_1]\ldots[s_n]}_{[s]}$ for the set of all subsort-overloaded operators f for these components. Given $(f : s_1 \ldots s_n \to s) \in f^{[s_1]\ldots[s_n]}_{[s]}$ define:

$$(f : s_1 \ldots s_n \to s)\downarrow = \{(f : s'_1 \ldots s'_n \to s') \in f^{[s_1]\ldots[s_n]}_{[s]} \mid s'_1 \ldots s'_n s' < s_1 \ldots s_n s\}.$$

as its set of *strictly smaller typings*. Define: $\Sigma_0 = \{(f : s_1 \ldots s_n \to s) \in \Sigma \mid (f : s_1 \ldots s_n \to s)\downarrow = \emptyset\}$, and, inductively, $\Sigma_{n+1} = \{(f : s_1 \ldots s_n \to s) \in \Sigma \mid (f : s_1 \ldots s_n \to s)\downarrow \subseteq \Sigma_n\}$. Because of the finiteness of Σ, we get a fixpoint $\Sigma_k = \Sigma_{k+1} = \Sigma$, giving us the above-mentioned telescope. Note that regularity of Σ_n, $n \geq 0$, follows easily by construction from the regularity of Σ. Furthermore, for any $t \in T_{\Sigma_n}(X)$ we have $ls_{\Sigma_n}(t) = ls_\Sigma(t)$. We will define a telescope $\Sigma^\#_0 \subseteq \Sigma^\#_1 \subseteq \ldots \Sigma^\#_{k-1} \subseteq \Sigma^\#$ that closely mirrors that of Σ. First of all, note that the map $atoms : S \to \mathcal{P}(S^\#)$ naturally extends to a map on strings, $atoms : S^* \to \mathcal{P}((S^\#)^*)$ by defining: $atoms(\epsilon) = \{\epsilon\}$, and $atoms(sw) = \{s'w' \mid s' \in atoms(s) \wedge w' \in atoms(w)\}$. Note also that the mapping $(f : s_1 \ldots s_n \to s) \mapsto s_1 \ldots s_n$ defines a function $arity : \Sigma \to S^*$. Define $\Sigma^\#_0 = \{(f : w \to s^\bullet) \mid (f : s_1 \ldots s_n \to s) \in \Sigma_0,\ w \in atoms(s_1 \ldots s_n)\}$, where $s^\bullet =$ **if** s atomic **then** s **else** $s^\#$ **fi**. Then

define $\Sigma^{\#}_{n+1}$ inductively as follows: $\Sigma^{\#}_{n+1} = \Sigma^{\#}_n \cup \{(f : w \to s^{\#}) \mid (f : s_1 \dots s_n \to s) \in \Sigma_{n+1} - \Sigma_n,\ s \textit{ irredundant},\ w \in \textit{atoms}(s_1 \dots s_n) - \{\textit{arity}(f : w' \to s') \mid (f : w' \to s') \in \Sigma^{\#}_n\}\}$. If $\Sigma_k = \Sigma$, we define $\Sigma^{\#}_k = \Sigma^{\#}$ and obtain a telescope $\Sigma^{\#}_0 \subseteq \Sigma^{\#}_1 \subseteq \dots \Sigma^{\#}_{k-1} \subseteq \Sigma^{\#}$ as claimed.

The main properties of the $\Sigma \mapsto \Sigma^{\#}$ transformation are as follows:

Theorem 3. *Let Σ satisfy the above assumptions. Then:*

1. *$\Sigma^{\#}$ is sensible*
2. *for $s, s' \in S^{\#}$, $s \neq s' \Rightarrow T_{\Sigma^{\#},s} \cap T_{\Sigma^{\#},s'} = \emptyset$*
3. *for each $s \in S$, $T_{\Sigma,s} = \biguplus_{s' \in \textit{atoms}(s)} T_{\Sigma^{\#},s'}$*
4. *$t \in T_{\Sigma} \wedge ls_{\Sigma}(t) = s \Leftrightarrow t \in T_{\Sigma^{\#}} \wedge ls_{\Sigma^{\#}}(t) = s^{\bullet}$.*

3.1 Variations on the $\Sigma^{\#}$ Theme

Several signatures closely related to Σ and $\Sigma^{\#}$ are also very useful. The most obvious is their union $\Sigma \cup \Sigma^{\#}$, with set of operators the set-theoretic union $\Sigma \cup \Sigma^{\#}$ and poset of sorts $(S \cup S^{\#}, (\leq \cup <^{\#})^*)$, with $\leq$ the order in $(S, \leq)$, and $<^{\#} = \{(s^{\#}, s) \mid s \textit{ nonatomic and irredundant}\}$. $\Sigma \cup \Sigma^{\#}$ is even more intuitive than $\Sigma^{\#}$, because it *refines* Σ into a richer semantics-preserving signature by just adding to it the new atoms $s^{\#}$, so that now the least sort of any ground term t will always be an atomic sort. This means that we have *sharpened* the typing of any such t as much as possible, which is the reason for the $\Sigma^{\#}$ notation. Note that we have subsignature inclusions $J : \Sigma \hookrightarrow \Sigma \cup \Sigma^{\#}$ and $J' : \Sigma^{\#} \hookrightarrow \Sigma \cup \Sigma^{\#}$. Furthermore, $\Sigma \cup \Sigma^{\#}$ enjoys very good properties, which make it an initial-semantics-preserving enrichment of both Σ and $\Sigma^{\#}$:

Lemma 1. *$\Sigma \cup \Sigma^{\#}$ is regular, $T_{\Sigma \cup \Sigma^{\#}} \mid_J = T_{\Sigma}$, and $T_{\Sigma \cup \Sigma^{\#}} \mid_{J'} = T_{\Sigma^{\#}}$.*

Two other useful signatures are $\Sigma^{\#}_{\top}$ and $\Sigma^{\#}_c$. $\Sigma^{\#}_{\top}$ is an order-sorted signature with operations those in $\Sigma^{\#}$ and with sorts $S_{\top} \cup S^{\#}$, where $S_{\top} = \{\top_{[s]} \mid [s] \in \hat{S}\}$ is the set of top sorts of each connected component in $(S, \leq)$. Its order is defined as the identity relation on $S^{\#} \cup S_{\top}$, plus the subsort inclusions $s' \leq \top_{[s]}$ for each $s' \in \textit{atoms}(\top_{[s]})$. We have a subsignature inclusion $K : \Sigma^{\#}_{\top} \hookrightarrow \Sigma \cup \Sigma^{\#}$. Reasoning as in Lemma 1 it is easy to show that $T_{\Sigma \cup \Sigma^{\#}} \mid_K = T_{\Sigma^{\#}_{\top}}$.

$\Sigma^{\#}_c$ is a *many-sorted version* of $\Sigma^{\#}_{\top}$. Its set of sorts is $S_{\top} \cup S^{\#}$, but now $s \leq s'$ iff $s = s'$. The operations of $\Sigma^{\#}_c$ are those of $\Sigma^{\#}$ plus the coercion operators $\{c : s' \to \top_{[s]} \mid [s] \in \hat{S},\ \ s' \in \textit{atoms}(\top_{[s]}) - \{\top_{[s]}\}\}$, which mimic the subsort inclusions $s' < \top_{[s]}$ in $\Sigma^{\#}_{\top}$. We then have a signature morphism $H : \Sigma^{\#}_c \to \Sigma^{\#}_{\top}$ that is the identity on sorts and on the operators in $\Sigma^{\#}$ and maps each coercion $c : s' \to \top_{[s]}$ to the term $x_1{:}s'$. The following diagram summarizes this section:

$$(\ddagger) \quad \Sigma \overset{J}{\hookrightarrow} \Sigma \cup \Sigma^{\#} \overset{K}{\hookleftarrow} \Sigma^{\#}_{\top} \overset{H}{\leftarrow} \Sigma^{\#}_c$$

4 Equational Formulas in Initial Order-Sorted Algebras

The main goal of this section is to *reduce* the validity of equational first-order formulas in an initial *order-sorted* algebra to the validity of semantically equivalent formulas in an initial *many-sorted* algebra. The main idea of this reduction is to exploit diagram (‡) at the end of Section 3.1, which begins with an order-sorted signature Σ and ends with a many-sorted signature $\Sigma^{\#}_{c}$. Like Alice in Wonderland's Cheshire cat's smile, all order-sorted features vanish in the passage from Σ to $\Sigma^{\#}_{c}$. This reduction seems useful for at least two reasons:

1. Its provides a new, very simple proof of the *decidability* of first-order formulas in initial order-sorted algebras. A non-trivial proof of such a decidability result goes back to [4,5], but it requires quite complex formulas and formula transformations involving sort constraints based on quite general sort expressions, whose semantics is defined using tree automata.
2. The reduction-based proof given here provides a useful new *transfer principle*, by which problems with a perhaps unclear solution at the order-sorted level can be reduced to problems having a clear solution at the many-sorted level. For example, as further explained in Section 5, the puzzling anomaly about pattern operations in initial order-sorted algebras discussed in the Introduction has a systematic solution thanks to this transfer principle.

The main idea of the reduction is to assign to each first-order sentence φ in the language of a finite and regular order-sorted signature Σ a corresponding sentence $\varphi^{\#}_{c}$ in the language of the many-sorted signature $\Sigma^{\#}_{c}$, and then prove that we have an equivalence $T_{\Sigma} \models \varphi \;\Leftrightarrow\; T_{\Sigma^{\#}_{c}} \models \varphi^{\#}_{c}$. To obtain such an equivalence we make our way from T_{Σ} and φ to $T_{\Sigma^{\#}_{c}}$ and $\varphi^{\#}_{c}$ by moving from left to right along the diagram (‡). Since some of the steps in this sequence of signature morphisms are easy consequences of the equivalence (†) at the end of Section 2, we can quickly get such easy equivalences out of the way. Indeed, since J is a subsignature inclusion, it is the identity on formulas, and since by Lemma 1 we have the equality $T_{\Sigma \cup \Sigma^{\#}}|_{J} = T_{\Sigma}$, (†) applied to J gives us the equivalence $T_{\Sigma} \models \varphi \;\Leftrightarrow\; T_{\Sigma \cup \Sigma^{\#}} \models \varphi$. On the leftmost side, (†) gives us the equivalence $T_{\Sigma^{\#}_{\top}} \models H(\varphi^{\#}_{c}) \;\Leftrightarrow\; T_{\Sigma^{\#}_{\top}}|_{H} \models \varphi^{\#}_{c}$. The interesting twist, however, in that the unique $\Sigma^{\#}_{c}$-homomorphism $h : T_{\Sigma^{\#}_{c}} \to T_{\Sigma^{\#}_{\top}}|_{H}$ from the initial $\Sigma^{\#}_{c}$-algebra $T_{\Sigma^{\#}_{c}}$ is obviously the identity on the sorts $S^{\#}$ and maps each term $c(t) \in T_{\Sigma^{\#}_{c}, \top_{[s]}}$ to the term $t \in T_{\Sigma^{\#}_{\top}, \top_{[s]}}$. That is, h is *bijective*, and therefore a $\Sigma^{\#}_{c}$*-isomorphism* $h : T_{\Sigma^{\#}_{c}} \cong T_{\Sigma^{\#}_{\top}}|_{H}$, which gives us the equivalence $T_{\Sigma^{\#}_{\top}}|_{H} \models \varphi^{\#}_{c} \;\Leftrightarrow\; T_{\Sigma^{\#}_{c}} \models \varphi^{\#}_{c}$. Therefore, stringing these last two equivalences together, we get the equivalence $T_{\Sigma^{\#}_{\top}} \models H(\varphi^{\#}_{c}) \;\Leftrightarrow\; T_{\Sigma^{\#}_{c}} \models \varphi^{\#}_{c}$. We will then be done proving our desired equivalence $T_{\Sigma} \models \varphi \;\Leftrightarrow\; T_{\Sigma^{\#}_{c}} \models \varphi^{\#}_{c}$ if we can define a mapping $\varphi \mapsto \varphi^{\#}$ such that $H(\varphi^{\#}_{c}) = \varphi^{\#}$ and we show an equivalence $T_{\Sigma \cup \Sigma^{\#}} \models \varphi \;\Leftrightarrow\; T_{\Sigma^{\#}_{\top}} \models \varphi^{\#}$.

What makes the mapping $\varphi \mapsto \varphi^{\#}$ not entirely obvious is that $\Sigma^{\#}_{\top}$ has considerably fewer sorts than the plentiful $\Sigma \cup \Sigma^{\#}$. In particular, we have to

find a way to express equations and quantifiers involving variables with sorts of $\Sigma \cup \Sigma^\#$ not present in $\Sigma^\#_\top$ in the poorer language of $\Sigma^\#_\top$. The key idea for this is to observe that every ground $\Sigma \cup \Sigma^\#$-term has an atomic least sort in $S^\#$, and that, by Theorem 3–(3) and Lemma 1, we have the equality $T_{\Sigma\cup\Sigma^\#,s} = \biguplus_{s' \in atoms(s)} T_{\Sigma^\#,s'}$. Therefore, abbreviating $t \in T_{\Sigma\cup\Sigma^\#,s}$ to $t : s$, we have, $t : s \;\Leftrightarrow \bigvee_{s' \in atoms(s)} t : s'$, which is a property expressible in the language of $\Sigma^\#_\top$. Here is now the detailed mapping $\varphi \mapsto \varphi^\#$ using these ideas. Without loss of generality we may assume φ in prenex form, that is, $\varphi = Q\varphi_0$, with Q a sequence of quantifiers and φ_0 quantifier-free. The mapping $\varphi \mapsto \varphi^\#$ decomposes into a mapping $\varphi_0 \mapsto \varphi^\#_0$ for the quantifier-free part and a mapping for the quantifiers.

We first need some notation. $\overline{x{:}s}$ abbreviates a sequence of variables $x_1 : s_1, \ldots, x_n{:}s_n$. We can always decompose the free variables of φ_0 as $fvars(\varphi_0) = \overline{x{:}s}, \overline{y{:}p}$, with $\overline{x{:}s}$ variables having non-atomic sorts, and $\overline{y{:}p}$ variables having atomic sorts. Also, if $\overline{x{:}s} = x_1 : s_1, \ldots, x_n : s_n$, then $\overline{x{:}s}_\top$ denotes the variables $\overline{x{:}s}_\top = x_1 : \top_{[s_1]}, \ldots, x_n : \top_{[s_n]}$. In the same spirit, $\overline{x{:}s} = \bar{t}$ abbreviates the *conjunction of equations* $x_1{:}s_1 = t_1 \wedge \ldots \wedge x_n{:}s_n = t_n$, and $\{\overline{x{:}s} = \bar{t}\}$ abbreviates the *substitution* $\{x_1 : s_1 \mapsto t_1, \ldots, x_n : s_n \mapsto t_n\}$. Given variables $\overline{x{:}s}$ with sorts in S, let $Spec(\overline{x{:}s}, S^\#)$, called the set of $S^\#$*-specializations* of $\overline{x{:}s}$, be the set $Spec(\overline{x{:}s}, S^\#) = \{\overline{x{:}s} = \overline{z{:}q} \mid |\overline{x{:}s}| = |\overline{z{:}q}| \wedge q_i \in atoms(s_i), 1 \leq i \leq |x{:}s|\}$, where $|\overline{x{:}s}|$ denotes the length of the sequence of variables $\overline{x{:}s}$. To avoid variable capture we will always assume that the variables $\overline{z{:}q}$ are *fresh* variables, different for each $(\overline{x{:}s} = \overline{z{:}q}) \in Spec(\overline{x{:}s}, S^\#)$ and not appearing anywhere else. Viewed as a substitution $\{\overline{x{:}s} = \overline{z{:}q}\}$, each specialization $\overline{x{:}s} = \overline{z{:}q}$ is just a variable mapping lowering the sort s_i of each x_i to a sort $q_i \in atoms(s_i)$ for z_i. We can now define the mapping $\varphi_0 \mapsto \varphi^\#_0$ —where $fvars(\varphi_0) = \overline{x{:}s}, \overline{y{:}p}$, with $\overline{x{:}s}$ variables having non-atomic sorts, and $\overline{y{:}p}$ variables having atomic sorts— as follows:

$$\varphi^\#_0 = \bigvee_{(\overline{x{:}s} = \overline{z{:}q}) \in Spec(\overline{x{:}s}, S^\#)} (\exists \overline{z{:}q})\ (\overline{x{:}s}_\top = \overline{z{:}q} \wedge (\varphi_0\{\overline{x{:}s} = \overline{z{:}q}\})).$$

Note that $fvars(\varphi^\#_0) = \overline{x{:}s}_\top, \overline{y{:}p}$. The semantic equivalence between φ_0 and $\varphi^\#_0$ can then be expressed as follows:

Lemma 2. *For φ_0 as above, $\alpha \in [\overline{x{:}s}_\top, \overline{y{:}p} \to T_{\Sigma^\#_\top}]$ satisfies $T_{\Sigma^\#_\top}, \alpha \models \varphi^\#_0$ iff there exists $\beta \in [\overline{x{:}s}, \overline{y{:}p} \to T_\Sigma]$ such that $\alpha = \beta \circ \{\overline{x{:}s}_\top = \overline{x{:}s}\}$ and $T_\Sigma, \beta \models \varphi_0$.*

Since $\varphi = Q\varphi_0$, to define $\varphi^\#$ we still need to deal with the quantifiers Q. This is done inductively for each individual quantifier as follows. If $s \in S \cap (S^\# \cup S_\top)$, then $((\forall x{:}s)\ \varphi)^\# = (\forall x{:}s)\ \varphi^\#$, and $((\exists x{:}s)\ \varphi)^\# = (\exists x{:}s)\ \varphi^\#$. Otherwise, let $atoms(s) = \{q_1, \ldots, q_k\}$, then, $((\forall x{:}s)\ \varphi)^\# = (\forall x{:}\top_{[s]})\ (((\exists \overline{z{:}q})\ \bigvee_{i=1}^k x{:}\top_{[s]} = z_i{:}q_i) \Rightarrow \varphi^\#)$, and $((\exists x{:}s)\ \varphi)^\# = (\exists x{:}\top_{[s]}, \overline{z{:}q})\ (\bigvee_{i=1}^k x{:}\top_{[s]} = z_i{:}q_i) \wedge \varphi^\#)$.

The key syntactic invariant maintained by this translation is of course that if $fvars(\varphi) = \overline{x{:}s}, \overline{y{:}p}$, then $fvars(\varphi^\#) = \overline{x{:}s}_\top, \overline{y{:}p}$. And the key semantic invariant is that for each $\alpha \in [\overline{x{:}s}_\top, \overline{y{:}p} \to T_{\Sigma^\#_\top}]$ we have $T_{\Sigma^\#_\top}, \alpha \models \varphi^\#$ iff there exists $\beta \in [\overline{x{:}s}, \overline{y{:}p} \to T_\Sigma]$ such that $\alpha = \beta \circ \{\overline{x{:}s}_\top = \overline{x{:}s}\}$ and $T_\Sigma, \beta \models \varphi$. For quantifier-free formulas this has already been proved in Lemma 2. That this remains true

after each quantification step is easy to check and left to the reader: indeed, the above treatment of quantifiers is analogous to how in set theory we restrict quantifiers ranging over all sets to quantifiers ranging over a given set A by defining $(\forall x \in A)\ \varphi = (\forall x)\ (x \in A \Rightarrow \varphi)$, and $(\exists x \in A)\ \varphi = (\exists x)\ (x \in A \wedge \varphi)$. Our treatment is not just analogous, but in fact a special case: we have just captured $x \in T_{\Sigma,s}$ by the formula $(\exists \overline{z{:}q})\ \bigvee_{i=1}^{k} x = z_i : q_i$. Therefore, for any sentence φ (i.e., $\mathit{fvars}(\varphi) = \emptyset$) we get $T_\Sigma \models \varphi \Leftrightarrow T_{\Sigma^\#_\top} \models \varphi^\#$.

To close all the proof steps along the Cheshire cat's sequence (‡) we need to define the formula $\varphi^\#_c$ such that $H(\varphi^\#_c) = \varphi^\#$. We can get $\varphi^\#_c$ from $\varphi^\#$ as follows. Since $\Sigma^\#_\top$ and $\Sigma^\#_c$ have the same sorts, the variables and quantifiers in $\varphi^\#$ and $\varphi^\#_c$ stay the same. We just replace each equation $u = v$ appearing somewhere in $\varphi^\#$ by the equation $c(u) = c(v)$ at the same position in $\varphi^\#_c$, unless: (i) $\top_{[s]}$ is atomic (then $u = v$ is left unchanged), or (ii) $\top_{[s]}$ is non-atomic and either u or v are variables of sort $\top_{[s]}$, which are then left unchanged. This gives us the desired semantic equivalence $T_\Sigma \models \varphi \Leftrightarrow T_{\Sigma^\#_c} \models \varphi^\#_c$.

Since both the technical report version [15] of Maher's paper [14], and the disunification paper by Comon and Lescanne [3] prove that the first-order theory of a *many-sorted* initial algebra T_Ω is *decidable* —i.e., that there is an algorithm to decide for any formula ϕ whether $T_\Omega \models \phi$ holds or not— we then get as a corollary of the above equivalence the following theorem,[1] already known since [4,5], but now obtained in a different way and with a considerably simpler proof:

Theorem 4. *Let Σ be a finite and regular order-sorted signature. For any first-order formula $\varphi \in \mathit{Form}(\Sigma)$ the validity problem $T_\Sigma \models \varphi$ is decidable.* □

5 Pattern Operations in Initial Order-Sorted Algebras

Given an order-sorted signature Σ, by a *Σ-pattern* we just mean a term $t \in T_\Sigma(X)$, where we assume X_s countably infinite for each sort $s \in S$. We call t a pattern to emphasize that t is a symbolic description of a *language*, namely the set $[\![t]\!] = \{t\sigma \mid \sigma \in [X \to T_\Sigma]\}$ of its *ground instances*. Similarly, a finite set of patterns $\{t_1, \ldots, t_n\}$ is a symbolic description of the language $[\![t_1, \ldots, t_n]\!] = [\![t_1]\!] \cup \ldots \cup [\![t_n]\!]$. A language need not be a set of strings. Since strings are just a special case of trees, it can be a *tree language*, that is, a subset $L \subseteq T_\Sigma$ for some Σ. Therefore, $\mathcal{P}(T_\Sigma)$ is the set of all Σ-tree languages, and we have a function

$$[\![\_]\!] : \mathcal{P}_{\mathit{fin}}(T_\Sigma(X)) \longrightarrow \mathcal{P}(T_\Sigma) : \{t_1, \ldots, t_n\} \mapsto [\![t_1, \ldots, t_n]\!]$$

sending each finite set of patterns to its associated language. Call a language $L \subseteq T_\Sigma$ a *pattern language* iff $L = [\![t_1, \ldots, t_n]\!]$ for some finite set of patterns $\{t_1, \ldots, t_n\}$. The most obvious question is that of *representability*: which languages $L \subseteq T_\Sigma$ are pattern languages, i.e., can be symbolically *represented* by

[1] Theorem 4 holds for Σ finite and regular because any such Σ can be transformed into a semantically equivalent signature with no ad-hoc overloading (by symbol renaming) and with each connected component having a top sort (added when missing).

some $\{t_1, \ldots, t_n\}$? Pattern languages are closed under finite unions by construction. Are they closed under finite intersections? Obviously *yes*, since, by distributivity we can reduce the problem to the intersection of two patterns $[\![u]\!] \cap [\![v]\!]$, and we have $[\![u]\!] \cap [\![v]\!] = [\![u\sigma_1]\!] \cup \ldots \cup [\![u\sigma_n]\!]$, where $\{\sigma_1, \ldots, \sigma_n\} = DUnif_{\Sigma}(u, v)$, the set of *most general disjoint order-sorted unifiers* of u and v in Σ [17]; that is, before unifying u and v, we rename v if necessary to make its variables disjoint from those of u. Are T_Σ and $\emptyset$ pattern languages? Yes: $\emptyset = [\![\emptyset]\!]$, and $T_\Sigma = [\![x_1 : \top_{[s_1]}, \ldots, x_k : \top_{[s_k]}]\!]$, where $\hat{S} = \{[s_1], \ldots, [s_k]\}$. So, the ony missing Boolean operation is *complement*. But since complement and difference are expressible in terms of each other: $\overline{A} = \top - A$, and $A - B = A \cap \overline{B}$, we can rephrase the question thus: are pattern languages closed under differences? In general they are *not*. For example, for Σ unsorted and having a constant a and a binary f, the language $[\![f(x, y)]\!] - [\![f(z, z)]\!]$ is *not* a pattern language (see Prop. 4.5 in [13]). However, in the unsorted case (see Corollary in pg. 314, [13]) $[\![t_1, \ldots, t_n]\!] - [\![t'_1, \ldots, t'_m]\!]$ *is* a pattern language when the t_i and the t'_j are *linear* terms —have no repeated variables— and more general cases than just sets of linear patterns also yield differences that are pattern languages [13,12,19,18,7].

Since all other Boolean operations are already taken care of, all we need is a way of symbolically defining the *difference* $\{t_1, \ldots, t_n\} - \{t'_1, \ldots, t'_m\}$ of two finite sets of *order-sorted* patterns whenever this represents a pattern language. As illustrated by the example in the Introduction, if we insist on remaining in the given signature Σ this cannot be done, even for sets of linear patterns. However, we can use the $\Sigma \mapsto \Sigma^{\#}$ transformation and the *transfer principle* from order-sorted problems to many-sorted ones discussed in Section 4 to obtain a solution based on the following two simple observations:

1. As *sets* (not as algebras) we have $T_\Sigma = T_{\Sigma^{\#}}$.
2. For any *order-sorted* pattern $t \in T_\Sigma(X)$ we have the *language equality* $[\![t]\!] = \bigcup_{(\overline{x:s} = \overline{z:q}) \in Spec(\overline{x:s}, S^{\#})} [\![t\{\overline{x:s} = \overline{z:q}\}]\!]$, where $\overline{x:s} = fvars(t)$.

where both (1) and (2) are simple corollaries of Theorem 3. This then yields a straightforward way of representing a difference of finite sets of *order-sorted* Σ-patterns $\{t_1, \ldots, t_n\} - \{t'_1, \ldots, t'_m\}$ as a difference of finite sets of *many-sorted* $\Sigma^{\#}$-patterns: we just replace each t_i (resp t'_j) by the finite set of many-sorted $\Sigma^{\#}$-patterns $\{t_i\{\overline{x:s} = \overline{z:q}\} \mid (\overline{x:s} = \overline{z:q}) \in Spec(\overline{x:s}, S^{\#})\}$, where $\overline{x:s} = fvars(t_i)$. For the example in the Introduction, this method transforms the order-sorted symbolic difference $\{x{:}B\} - \{y{:}A\}$ into the many-sorted symbolic difference: $\{x{:}A, z{:}B^{\#}\} - \{y{:}A\}$.

Since —with the possible exception of the treatment of *finite sorts* (see below), which warrants an extension of the unsorted algorithms— the unsorted algorithms for computing the symbolic difference of two sets of patterns have a straightforward generalization to the many-sorted case, we can just use the above reduction to the many-sorted case and many-sorted versions of the difference algorithms in [13,19,18,7] to solve the problem of computing when possible the symbolic difference of order-sorted patterns $\{t_1, \ldots, t_n\} - \{t'_1, \ldots, t'_m\}$ as a finite set of (many-sorted) patterns.

But is this the best we can do? There can be some practical limitations, both in performance and at the representational level. For order-sorted signatures with rich type structures a set $atoms(s)$ may have a considerable number of sorts in $S^{\#}$, so that the sets $\{t_i\{\overline{x:s} = \overline{z:q}\} \mid (\overline{x:s} = \overline{z:q}) \in Spec(\overline{x:s}, S^{\#})\}$ for each t_i (resp. t'_i) can become quite big, affecting performance. It also means that the representation of the *solutions* to symbolic difference problems, besides being possibly quite big, may also be more *verbose* than necessary. For example, in the signature of the Introduction, we can compute the *order-sorted* symbolic difference $\{x{:}B\} - \{b\} = \{f(y{:}B), a\}$, which is shorter and more intuitive than the equivalent many-sorted representation $\{x{:}B\} - \{b\} = \{f(z{:}B^{\#}), f(z'{:}A), a\}$.

We present below an attractive alternative, namely, an *order-sorted* algorithm for computing symbolic differences $\{t_1, \ldots, t_n\} - \{t'_1, \ldots, t'_m\}$ in the extended order-sorted signature $\Sigma \cup \Sigma^{\#}$ that does not require any transformation of the original problem and can significantly overcome the above limitations by yielding simpler and shorter representations and better performance (see Appendix C).

Let us describe this algorithm. First of all, thanks to the Boolean equation $(A \cup B) - C = (A - C) \cup (B - C)$, we can decompose $\{t_1, \ldots, t_n\} - \{t'_1, \ldots, t'_m\}$ as a union $\{t_1\} - \{t'_1, \ldots, t'_m\} \cup \ldots \cup \{t_n\} - \{t'_1, \ldots, t'_m\}$. Second, thanks to the Boolean equation $A - B = A - (A \cap B)$ we can reduce $\{t\} - \{t_1, \ldots, t_n\}$ to the equivalent symbolic expression $\{t\} - \{t\sigma \mid \sigma \in DUnif_{\Sigma}(t, t_1) \cup \ldots \cup DUnif_{\Sigma}(t, t_n)\}$. Thus, all our symbolic difference problems can be reduced to unions of problems of the form $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$ with $\sigma_1, \ldots, \sigma_n$ substitutions instantiating t. Our algorithm gives priority to the easier and frequently occurring cases, using the order-sorted extension of the more general algorithm of Lassez and Marriott [13] only when the simpler algorithms cannot be applied. We also exploit the fact that a sort s may be *finite* —i.e., $T_{\Sigma \cup \Sigma^{\#}, s}$ is a finite set— plus the decidability of sort finiteness to increase the successful difference cases. Specifically:

1. If $t, t\sigma_1, \ldots, t\sigma_n$ are all *linear* terms, we apply the inference rules below.
2. Otherwise, when $\sigma_1, \ldots, \sigma_n$ are all *linear*, i.e., $\sigma_i(x), \sigma_i(y)$ are linear terms not sharing any variables when $x \neq y$, we reduce to case (1) (Appendix B).
3. Otherwise, if σ_i is non-linear and $y{:}s$ occurs more than once either in $\sigma_i(x)$ or in $\sigma_i(x), \sigma_i(z)$, $x \neq z$, with s finite, $T_{\Sigma \cup \Sigma^{\#}, s} = \{u_1, \ldots, u_k\}$, then we replace the problem $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$ by the problem $\{t\} - \{t\sigma_1, \ldots, t\sigma_i\{y \mapsto u_1\}, \ldots, t\sigma_i\{y \mapsto u_k\}, \ldots, t\sigma_n\}$ and check again the new problem.
4. Outside cases (1)–(3) above, we invoke the order-sorted version of the algorithm in [13], which is more efficient than those in [19,18,7] and gives a full answer to difference problems $\{t\} - \{t_1, \ldots, t_n\}$, whereas those in [19,18,7] give a full answer to arbitrary Boolean combinations (see Appendix B).

In case all terms $t, t\sigma_1, \ldots, t\sigma_n$ are linear, the following rewrite rules are applied:

1. $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\} \to (\{t\} - \{t\sigma_1\}) \cap \ldots \cap (\{t\} - \{t\sigma_n\})$
2. $\{t\} - \emptyset \to \{t\}$
3. $\{f(t_1, \ldots, t_n)\} - \{f(t_1\sigma, \ldots, t_n\sigma)\} \to \{f(t_1, \ldots, u, \ldots, t_n) \mid u \in (\{t_i\} - \{t_i\sigma\}), 1 \leq i \leq n\}$, where $fvars(u)$ are fresh variables.
4. $\{x{:}s\} - \{y{:}s'\} \to \{z_1{:}q_1, \ldots, z_k{:}q_k\}$, where $\{q_1, \ldots, q_k\} = atoms(s) - atoms(s')$

5. $\{x:s\} - \{f(t_1,\ldots,t_n)\} \to \{\overline{z{:}q}\} \cup \bigcup\{\{x_p:p\} - \{f(t_1,\ldots,t_n)\{\rho\} \mid \rho \in Spec(Y,S^\#),\ p = ls_{\Sigma^\#}(f(t_1,\ldots,t_n)\{\rho\})\} \mid p \in atoms(s) \cap atoms(f(t_1,\ldots,t_n))\}$ *if* $s \notin S^\#$, where $Y = fvars(f(t_1,\ldots,t_n))$, $\overline{z{:}q} = z_1{:}q_1,\ldots,z_k{:}q_k$, $\{q_1,\ldots,q_k\} = atoms(s) - atoms(f(t_1,\ldots,t_n))$, and $atoms(f(t_1,\ldots,t_n)) = \{ls_{\Sigma^\#}(f(t_1,\ldots,t_n)\{\rho\}) \mid \rho \in Spec(Y,S^\#)\}$.
6. $\{x{:}s\} - \{f(t_1,\ldots,t_n)\} \to \{u \mid u \in Pat(s) - \{f(\overline{x{:}s})\}\} \cup \{f(\overline{x{:}s})\} - \{f(t_1,\ldots,t_n)\}$ *if* $s = ls_{\Sigma^\#}(f(t_1,\ldots,t_n)) \in S^\#$, where $\overline{x{:}s} = x_1{:}s_1,\ldots,x_n{:}s_n$, $s_i = ls_{\Sigma^\#}(t_i)$, and $Pat(s) = \{g(x_1{:}s_1,\ldots,x_n{:}s_n) \mid g : s_1 \ldots s_n \to s \in \Sigma^\#\}$.

The correctness of these rules and of the algorithm is proved in Appendix B.

What advantages do we gain from this algorithm? Quite substantial ones to reason effectively about languages. Let $LT_\Sigma(X) \subseteq T_\Sigma(X)$ denote the set of *linear* terms in $T_\Sigma(X)$. Note that if $u \in LT_\Sigma(X)$ then $[\![u]\!]$ is a *regular* tree language. This follows from order-sorted signatures being tree automata, plus the regular expression fact that if $L_1,\ldots,L_n$ are regular languages, then $f(L_1,\ldots,L_n)$ is a regular language. Also, $\mathcal{P}_{fin}(LT_{\Sigma\cup\Sigma^\#}(X))$ is closed under symbolic: (i) unions; (ii) intersections, because disjoint unifiers of linear terms are linear; and (iii) differences, since rules (1)–(6) preserve linearity of terms. Furthermore, given $\{t_1,\ldots,t_n\},\{t'_1,\ldots,t'_m\} \in \mathcal{P}_{fin}(LT_{\Sigma\cup\Sigma^\#}(X))$ we can use pattern differences to *decide* whether $[\![\{t_1,\ldots,t_n\}]\!] = [\![\{t'_1,\ldots,t'_m\}]\!]$. Indeed, $[\![\{t_1,\ldots,t_n\}]\!] = [\![\{t'_1,\ldots,t'_m\}]\!] \Leftrightarrow \{t_1,\ldots,t_n\} \equiv \{t'_1,\ldots,t'_m\}$, where, by definition, $\{t_1,\ldots,t_n\} \equiv \{t'_1,\ldots,t'_m\}$ iff $\{t_1,\ldots,t_n\} - \{t'_1,\ldots,t'_m\} = \emptyset$ and $\{t'_1,\ldots,t'_m\} - \{t_1,\ldots,t_n\} = \emptyset$. By the homomorphism theorem for Boolean algebras, this means that $[\![\_]\!]$ defines an *injective homomorphism of Boolean algebras* $[\![\_]\!] : \mathcal{P}_{fin}(LT_{\Sigma\cup\Sigma^\#}(X))/\equiv \to \mathcal{P}(T_\Sigma)$. This is as good as it gets, since $\mathcal{P}_{fin}(LT_{\Sigma\cup\Sigma^\#}(X))/\equiv$ is a *computable* Boolean algebra, where all operations become effective. This offers an attractive, simpler alternative to tree automata to effectively compute Boolean operations on linear pattern languages in a symbolic way.

6 Implementation and Experiments

The algorithms described in this paper are highly *reflective*. That is, they are *parametric* on signatures Σ and perform meta-level operations on signatures and Σ-terms, such as order-sorted unification, matching, sort comparisons, and so on, to ultimately compute pattern operations. Fortunately, many of these auxiliary meta-level operations are available, or can be easily programmed, in the Maude language through its reflective features using its `META-LEVEL` module [2]. In `META-LEVEL`, a signature Σ is meta-represented as a term $\overline{\Sigma}$ of sort `Module`, and a Σ-term t is meta-represented as a so-called *meta-term* $\overline{t}$ of sort `Term`.

Since: (i) Maude syntax at the meta-level essentially mirrors the syntax at the object level; and (ii) inference rules such as above rules (1)–(6) can be directly expressed as rewrite rules operating on meta-terms, the *representational distance* between the theoretical description of the algorithm in Section 5 and its actual meta-level implementation in Maude is relatively short.

We have implemented in Maude the signature transformation $\Sigma \mapsto \Sigma^\#$ described in Section 3. The implementation essentially coincides with the tele-

scoping procedure described therein. The procedure takes a reflected signature $\overline{\Sigma}$ as an argument and proceeds by non-deterministically selecting an operator f in Σ which has not been processed and whose strictly smaller typings have all been processed. Using the signature transformation procedure we have also implemented the order-sorted pattern operation algorithms described in Section 5. The overall algorithm takes as arguments a reflected signature $\overline{\Sigma}$, a set of pre-computed ground meta-term sets inhabiting each finite sort, and a symbolic Boolean expression composed of meta-terms $\bar{t}$ representing Σ-term patterns using a mixfix syntax where `_U_` represents union, `_&_` represents intersection, and `_-_` represents difference. A set of Boolean equations first reduces each Boolean symbolic pattern expression to a *normal form* (essentially pushing conjunctions/differences down the expression tree). A normal form is then solved using an algorithm that, with some additional optimizations and small variations, deals with each symbolic difference problem according to the steps described in Section 5: the problem is first classified according to cases (1)–(4), iterating over the finite-sort transformation of case (3) if needed. Then, either the simpler algorithm for case (1) (essentially rules (1)–(6)), or its case (2) extension (see Appendix B), or the more general order-sorted extension of the Lassez-Marriott algorithm [13] are invoked. Finally, symbolic union and intersection operations are performed to obtain either: (i) a finite set of patterns if the algorithm computed a pattern language, or (ii) a Boolean expression containing some symbolic differences that do not denote pattern languages otherwise.

Additionally, we conducted experiments comparing our order-sorted pattern operations algorithm to its many-sorted reduction. To ground our discussion, we work in a module `COMPLEX-RAT` adapted from [10] that defines the complex numbers. We also fix a term set T by randomly selecting operators to generate terms upto depth 2. Then, given $(t_1, t_2) \in T^2$, we generate the pattern operation $[\![t_1]\!] - [\![t_2]\!]$, which we compute by both our order-sorted algorithm and the many-sorted reduction. Our experiments show that, on average, the many-sorted reduction requires about a 1,000 times as many rewrites as the order-sorted algorithm, with the median being 55 times as many rewrites. While not a proof, this presents a strong case that for (non-toy) examples, the order-sorted algorithm is more expressive (no input encoding, shorter output) and performant than its many-sorted cousin. For more details on our experiments see Appendix C.

7 Related Work and Conclusions

On pattern operations the most closely related work is [13,12,19,18,7] and references there. On equational formulas in initial algebras the most closely related work is [14,15,3,4,5] and references there. The relationships to work in both these areas have been discussed in detail in previous sections (see also Appendix B).

To conclude, we have shown that the untyped algorithms break down when performing the order-sorted pattern operations needed in current declarative languages, and shown that such operations can be defined using a signature transformation $\Sigma \mapsto \Sigma^{\#}$. We have also shown that this transformation yields

new insights and a new, quite simple proof of the known decidability of the first-order theory of an initial order-sorted algebra. The Introduction mentioned many applications of pattern operations. We illustrate a sufficient completeness one in Appendix D, but plan to work on many others and to further advance the current implementation to make it part of the Maude formal tool environment.

References

1. Alpuente, M., Escobar, S., Espert, J., Meseguer, J.: A modular order-sorted equational generalization algorithm. Inf. Comput. 235, 98–136 (2014)
2. Clavel, M., Durán, F., Eker, S., Meseguer, J., Lincoln, P., Martí-Oliet, N., Talcott, C.: All About Maude. Springer LNCS 4350 (2007)
3. Comon, H., Lescanne, P.: Equational problems and disunification. Journal of Symbolic Computation 7, 371–425 (1989)
4. Comon, H.: Equational formulas in order-sorted algebras. In: Proc. ICALP'90. LNCS, vol. 443, pp. 674–688. Springer (1990)
5. Comon, H., Delor, C.: Equational formulae with membership constraints. Inf. Comput. 112(2), 167–216 (1994)
6. Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification 1. Springer (1985)
7. Fernández, M.: Negation elimination in empty or permutative theories. J. Symb. Comput. 26(1), 97–133 (1998)
8. Futatsugi, K., Diaconescu, R.: CafeOBJ Report. World Scientific (1998)
9. Goguen, J., Burstall, R.: Institutions: Abstract model theory for specification and programming. Journal of the ACM 39(1), 95–146 (1992)
10. Goguen, J., Meseguer, J.: Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theoretical Computer Science 105, 217–273 (1992)
11. Goguen, J., Winkler, T., Meseguer, J., Futatsugi, K., Jouannaud, J.P.: Introducing OBJ. In: Software Engineering with OBJ: Algebraic Specification in Action, pp. 3–167. Kluwer (2000)
12. l. Lassez, J., Maher, M., Marriott, K.: Elimination of negation in term algebras. In: In Mathematical Foundations of Computer Science. pp. 1–16. Springer (1991)
13. Lassez, J.L., Marriott, K.: Explicit representation of terms defined by counter examples. J. Autom. Reasoning 3(3), 301–317 (1987)
14. Maher, M.J.: Complete axiomatizations of the algebras of finite, rational and infinite trees. In: Proc. LICS '88. pp. 348–357. IEEE Computer Society (1988)
15. Maher, M.J.: Complete axiomatizations of the algebras of finite, rational and infinite trees. Tech. rep., IBM T. J. Watson Research Center (1988)
16. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Proc. WADT'97. pp. 18–61. Springer LNCS 1376 (1998)
17. Meseguer, J., Goguen, J., Smolka, G.: Order-sorted unification, *Journal of Symbolic Computation*, 8, 383–413 (1989)
18. Pichler, R.: Explicit versus implicit representations of subsets of the Herbrand universe. Theor. Comput. Sci. 290(1), 1021–1056 (2003)
19. Tajine, M.: The negation elimination from syntactic equational formula is decidable. In: Proc. RTA-93. LNCS, vol. 690, pp. 316–327. Springer (1993)

A Proofs of Theorems and Lemmas in Sections 3–4

Proof of Theorem 3.

Proof. It is easy to prove by structural induction that any term in a sensible many-sorted signature has a unique sort, so that (2) follows from (1). For the same reason, (1) makes $\Sigma^{\#}$ trivially regular. Furthermore, from the definition of the function *atoms* and (4), we can easily obtain (3). So, we only need to prove (1) and (4). Since $\Sigma^{\#}$ is sensible iff each $\Sigma^{\#}_n$ is sensible, we can prove (1) by proving that $\Sigma^{\#}_n$ is sensible for each n by induction on n.
Base case: $n = 0$. We can prove $\Sigma^{\#}_0$ sensible by contradiction. Suppose that we have two typings $f : w^{\bullet} \to s^{\bullet}$ and $f : w^{\bullet} \to s'^{\bullet}$ in $\Sigma^{\#}_0$ with $s \neq s'$. By the definition of Σ_0, this can only happen if f is not a constant and we have two different subsort-overloaded typings $f : u \to s, f : u' \to s' \in \Sigma_0$ and $w^{\bullet} \in \mathit{atoms}(u) \cap \mathit{atoms}(u')$. By the definition of the function *atoms* this can only happen if we have $w \leq u, u'$. By regularity this requires the existence of $f : w' \longrightarrow s''$ in Σ with $w \leq w' \leq u, u'$ and $w's'' \leq ws, ws'$. By the definition of Σ_0 this can only happen if either $w's'' = ws$ and $ws < ws'$, or $w's'' = ws'$ and $ws' < ws$; but in either case we cannot have $f : u \to s, f : u' \to s' \in \Sigma_0$.
Induction Step. We assume that $\Sigma^{\#}_n$ is sensible and prove $\Sigma^{\#}_{n+1}$ sensible. Note that, by the definition of $\Sigma^{\#}_{n+1}$, $f : w^{\bullet} \to s^{\#} \in \Sigma^{\#}_{n+1} - \Sigma^{\#}_n$ iff there is an $f : u \to s \in \Sigma_{n+1} - \Sigma_n$ with s irredundant and $w^{\bullet} \in \mathit{atoms}(u) - \{\mathit{arity}(f : w'' \to s'') \mid (f : w'' \to s'') \in \Sigma^{\#}_n\}$. Therefore, since $\Sigma^{\#}_n$ is sensible, a failure of $\Sigma^{\#}_{n+1}$ being sensible can only happen with two different typings $f : w^{\bullet} \to s^{\#}, f : w^{\bullet} \to s'^{\#} \in \Sigma^{\#}_{n+1} - \Sigma^{\#}_n$. This means that we have two different subsort-overloaded $f : u \to s, f : u' \to s' \in \Sigma_{n+1} - \Sigma_n$ with $w^{\bullet} \in \mathit{atoms}(u) \cap \mathit{atoms}(u')$. By the definition of the function *atoms* this can only happen if we have $w \leq u, u'$. But then regularity requires the existence of $f : w' \to s'''$ with $w \leq w' \leq u, u'$ and $w's''' \leq us, u's'$, which forces $w^{\bullet} \in \mathit{atoms}(w')$. Since $w^{\bullet} \notin \{\mathit{arity}(f : w'' \to s'') \mid (f : w'' \to s'') \in \Sigma^{\#}_n\}\}$ and $w's''' \leq us, u's'$, we must have $f : w' \to s''' \in \Sigma_{n+1} - \Sigma_n$. But this then forces either $w's'' = us$ and $us < u's'$, or $w's'' = u's'$ and $u's' < us$, both contradicting $f : u \to s, f : u' \to s' \in \Sigma_{n+1} - \Sigma_n$.

The proof of (4) essentially reduces to proving the following lemma:

Lemma 3. *Let $u \leq w$ be words of equal length in S^* such that $f : w \to s' \in \Sigma$, and let $f : v \to s$ be the smallest possible typing in Σ with $u \leq v \leq w$ and $vs \leq ws'$. Then, $f : u^{\bullet} \to s^{\bullet} \in \Sigma^{\#}$.*

Proof. If $f : v \to s \in \Sigma_0$, this follows from the definition of $\Sigma^{\#}_0$. Suppose $f : v \to s \in \Sigma_{n+1} - \Sigma_n$. Since Σ has non-empty sorts, there must then be terms of smallest sort s, so that s is irredundant. Therefore, the only way in which we can fail to have $f : u^{\bullet} \to s^{\bullet} \in \Sigma^{\#}_{n+1}$ is by having some $f : u^{\bullet} \to s''^{\bullet} \in \Sigma^{\#}_n$. But this can only happen if there is an $f : v' \to s'' \in \Sigma_n$ with $u \leq v'$, which by regularity forces $v's'' \geq vs$, and, since $v's'' \neq vs$ makes $v's'' > vs$, which forces $f : v \to s \in \Sigma_{n-1}$, contradicting $f : v \to s \in \Sigma_{n+1} - \Sigma_n$. □

Using (1), which ensures that $ls_{\Sigma^\#}(t)$ is a well-defined function, we can now prove (4) by structural induction. To see the ($\Rightarrow$) implication, note that the result is trivial if t is a constant, so let $f(t_1,\ldots,t_n) \in T_\Sigma \wedge ls_\Sigma(f(t_1,\ldots,t_n)) = s$, with $ls_\Sigma(t_i) = s_i$, $1 \leq i \leq n$. By the induction hypothesis we then have $t_i \in T_{\Sigma^\#} \wedge ls_{\Sigma^\#}(t_i) = s_i^\bullet$, $1 \leq i \leq n$. Furthermore, $ls_\Sigma(f(t_1,\ldots,t_n)) = s$ and regularity imply that we have an $f : v \to s \in \Sigma$ with $u = s_1 \ldots s_n \leq v$ and vs smallest possible with this property. But then Lemma 3 ensures the existence of $f : u^\bullet \to s^\bullet \in \Sigma^\#$, proving $f(t_1,\ldots,t_n) \in T_{\Sigma^\#} \wedge ls_{\Sigma^\#}(f(t_1,\ldots,t_n)) = s$, as desired.
For the ($\Leftarrow$) implication, constants are again trivial. So let $f(t_1,\ldots,t_n) \in T_{\Sigma^\#} \wedge ls_{\Sigma^\#}(f(t_1,\ldots,t_n)) = s^\bullet$, with $ls_{\Sigma^\#}(t_i) = s_i^\bullet$, $1 \leq i \leq n$. The induction hypothesis then gives us $t_i \in T_\Sigma \wedge ls_\Sigma(t_i) = s_i$, $1 \leq i \leq n$. Since for $u = s_1 \ldots s_n$ we have $f : u^\bullet \to s^\bullet \in \Sigma^\#$, by the construction of $\Sigma^\#$ we must have some $f : w \to s' \in \Sigma$ with $u \leq w$, and if $f : v \to s''$ is the smallest possible typing in Σ with $u \leq v \leq w$ and $vs'' \leq ws'$, then Lemma 3 ensures the existence of $f : u^\bullet \to s''^\bullet \in \Sigma^\#$, and $\Sigma^\#$ sensible forces $s = s''$. Therefore, $f(t_1,\ldots,t_n) \in T_\Sigma \wedge ls_\Sigma(f(t_1,\ldots,t_n)) = s$, as desired. □

Proof of Lemma 1

Proof. Let $u \in (S \cup S^\#)^*$ be such that there is an $f : w \to s'$ in $\Sigma \cup \Sigma^\#$ and $u \leq w$. If $u = u'^\bullet$, then the construction of $\Sigma^\#$, Lemma 3, and the order $(\leq \cup <^\#)^*$ ensure that there is a smallest possible typing of the form $f : u'^{bullet} \to s^\bullet$. Otherwise, $u \notin (S^\#)^*$, say, $u = s_1 \ldots s_{i_1}^\bullet \ldots s_{i_k}^\bullet \ldots s_n$, with $0 \leq k < n$. But then the only $f : w \to s'$ in $\Sigma \cup \Sigma^\#$ with $u \leq w$ and those $f : w \to s'$ in Σ with $s_1 \ldots s_{i_1} \ldots s_{i_k} \ldots s_n \leq w$, for which there is one with smallest possible typing by the regularity of Σ. The identity $T_{\Sigma \cup \Sigma^\#} \mid_{J'} = T_{\Sigma^\#}$ follows easily from the fact that the atomic sorts of $\Sigma \cup \Sigma^\#$ are precisely the sorts in $S^\#$, and the only operators relating those sorts are exactly those in $\Sigma^\#$. Note also that it follows easily from Theorem 3 that, as sets of terms, we have $T_{\Sigma \cup \Sigma^\#} = T_\Sigma = T_{\Sigma^\#}$, and that for any term in that set we have $ls_{\Sigma \cup \Sigma^\#}(t) = ls_{\Sigma^\#}(t)$. Therefore, to prove $T_{\Sigma \cup \Sigma^\#} \mid_J = T_\Sigma$ we just have to show that for each $s \in S$ we have $T_{\Sigma \cup \Sigma^\#, s} = T_{\Sigma, s}$. But since in the order $(\leq \cup <^\#)^*$ the atoms below any $s \in S$ are precisely the set of sorts $atoms(s)$, the equality $ls_{\Sigma \cup \Sigma^\#}(t) = ls_{\Sigma^\#}(t)$ and (3) in Theorem 3 give us $T_{\Sigma \cup \Sigma^\#, s} = T_{\Sigma, s}$, as desired. □

Proof of Lemma 2

Proof. To see the ($\Rightarrow$) implication, note that $T_{\Sigma_\top^\#}, \alpha \models \varphi_0^\#$ iff there is an $(\overline{x:s} = \overline{z:q}) \in Spec(\overline{x:s}, S^\#)$ such that $T_{\Sigma_\top^\#}, \alpha \models (\exists \overline{z{:}q})\ (\overline{x{:}s}_\top = \overline{z{:}q} \wedge (\varphi_0\{\overline{x{:}s} = \overline{z{:}q}\}))$. For each $1 \leq i \leq |\overline{x:s}|$ this forces $\alpha(x_i{:}\top_{[s_i]}) \in T_{\Sigma^\#, q_i}$, and, since $q_i \in atoms(s_i)$, by Theorem 3–(3) we must have $\alpha(x_i{:}\top_{[s_i]}) \in T_{\Sigma, s_i}$. This means that there is a $\beta \in [\overline{x{:}s}, \overline{y{:}p} \to T_\Sigma]$ with $\beta \mid_{\overline{y:p}} = \alpha \mid_{\overline{y:p}}$, and $\beta(x_i{:}s_i) = \alpha(x_i{:}\top_{[s_i]})$, $1 \leq i \leq |\overline{x:s}|$, such that $\alpha = \beta \circ \{\overline{x{:}s}_\top = \overline{x{:}s}\}$. To see that $T_\Sigma, \beta \models \varphi_0$ it is enough to reason by structural induction on the Boolean structure of φ_0 and show that for each equation $u = v$ appearing in φ_0, assuming $k = |\overline{z{:}q}|$, we have the equivalence: $T_{\Sigma_\top^\#}, \alpha \mid_{\overline{y:p}} \cup \{z_1{:}q_1 \mapsto \alpha(x_1{:}\top_{[s_1]}), \ldots, z_k{:}q_k \mapsto \alpha(x_k{:}\top_{[s_k]})\} \models (u = v)\{\overline{x{:}s} =$

$\overline{z{:}q}\} \Leftrightarrow T_\Sigma, \beta \models u = v$. But this is trivial, since by the definition of β we have $u\{\overline{x{:}s} = \overline{z{:}q}\}(\alpha\,|_{\overline{y:p}} \cup \{z_1{:}q_1 \mapsto \alpha(x_1{:}\top_{[s_1]}), \ldots, z_k{:}q_k \mapsto \alpha(x_k{:}\top_{[s_k]})\}) = u\beta$, and $v\{\overline{x{:}s} = \overline{z{:}q}\}(\alpha\,|_{\overline{y:p}} \cup \{z_1{:}q_1 \mapsto \alpha(x_1{:}\top_{[s_1]}), \ldots, z_k{:}q_k \mapsto \alpha(x_k{:}\top_{[s_k]})\}) = v\beta$.

The ($\Leftarrow$) implication can also be reduced to the case $\varphi_0 = u = v$, with $\mathit{fvars}(u = v) = \overline{x{:}s}, \overline{y{:}p}$. For any $\beta \in [\overline{x{:}s}, \overline{y{:}p} \to T_\Sigma]$ such that $T_\Sigma, \beta \models u = v$, let $q_i = ls_\Sigma(\beta(x_i{:}s_i))^\bullet$, and let $\alpha = \beta \circ \{\overline{x{:}s}_\top = \overline{x{:}s}\}$. Then we have $u\{\overline{x{:}s} = \overline{z{:}q}\}(\alpha\,|_{\overline{y:p}} \cup \{z_1 : q_1 \mapsto \alpha(x_1 : \top_{[s_1]}), \ldots, z_k : q_k \mapsto \alpha(x_k : \top_{[s_k]})\}) = u\beta = v\beta = v\{\overline{x{:}s} = \overline{z{:}q}\}(\alpha\,|_{\overline{y:p}} \cup \{z_1 : q_1 \mapsto \alpha(x_1 : \top_{[s_1]}), \ldots, z_k : q_k \mapsto \alpha(x_k : \top_{[s_k]})\})$, and therefore $T_{\Sigma^\#_\top}, \alpha \models (\exists \overline{z{:}q})\ (\overline{x{:}s}_\top = \overline{z{:}q} \wedge (\varphi_0\{\overline{x{:}s} = \overline{z{:}q}\}))$, which proves $T_{\Sigma^\#_\top}, \alpha \models \varphi_0^\#$, as desired. □

B The Order-Sorted Symbolic Difference Algorithm

We can show the correctness of rules (1)–(6) for the linear patterns case by showing that each rule $\mathit{exp} \to \mathit{exp}'$ is *language-preserving*, i.e., that $[\![\mathit{exp}]\!] = [\![\mathit{exp}']\!]$. For rule (1) this follows from the Boolean equation $A - (B \cup C) = (A - B) \cap (A - C)$. For rule (2) this is trivial. For rule (3) this follows from the following set equalities: $[\![f(t_1, \ldots, t_n)]\!] - [\![f(t_1\sigma, \ldots, t_n\sigma)]\!] = \{f(t_1, \ldots, t_n)\gamma \mid \gamma \in [Y \to T_\Sigma]\} - \{f(t_1, \ldots, t_n)\gamma \mid \gamma \in [Y \to T_\Sigma] \wedge t_1\gamma \in [\![t_1\sigma]\!] \wedge \ldots \wedge t_n\gamma \in [\![t_n\sigma]\!]\} = \{f(t_1, \ldots, t_n)\gamma \mid \gamma \in [Y \to T_\Sigma] \wedge \neg(t_1\gamma \in [\![t_1\sigma]\!] \wedge \ldots \wedge t_n\gamma \in [\![t_n\sigma]\!])\} = \{f(t_1, \ldots, t_n)\gamma \mid \gamma \in [Y \to T_\Sigma] \wedge (t_1\gamma \notin [\![t_1\sigma]\!] \vee \ldots \vee t_n\gamma \notin [\![t_n\sigma]\!])\} = \{f(t_1, \ldots, t_n)\gamma \mid \gamma \in [Y \to T_\Sigma] \wedge t_1\gamma \in [\![t_1 - t_1\sigma]\!] \cup \ldots \cup \{f(t_1, \ldots, t_n)\gamma \mid \gamma \in [Y \to T_\Sigma] \wedge t_n\gamma \in [\![t_n - t_n\sigma]\!]\} = \bigcup_{u_1 \in (\{t_1\} - \{t_1\sigma\})} [\![f(u_1, \ldots, t_n)]\!] \cup \ldots \cup \bigcup_{u_n \in (\{t_n\} - \{t_n\sigma\})} [\![f(t_1, \ldots, u_n)]\!]$. For rule (4) it follows easily from (2)–(3) in Theorem 3. For rule (5) this follows again from (2)–(3) in Theorem 3, the fact that if $\biguplus_p A_p$ is a disjoint union and $B_p \subseteq A_p$, then $\biguplus_p A_p - \biguplus_p B_p = \biguplus_p (A_p - B_p)$, and the fact that $[\![f(t_1, \ldots, t_n)]\!] = \biguplus_{\rho \in \mathit{Spec}(Y, S^\#)} [\![f(t_1, \ldots, t_n)\{\rho\}]\!]$. In this case $A_p = [\![x_p : p]\!]$, and $B_p = [\![\{f(t_1, \ldots, t_n)\{\rho\} \mid \rho \in \mathit{Spec}(Y, S^\#),\ p = ls_{\Sigma^\#}(f(t_1, \ldots, t_n)\{\rho\})\}]\!]$. The correctness of rule (6) follows for each $s \in S^\#$ from the language equality $[\![x{:}s]\!] = \biguplus_{v \in \mathit{Pat}(s)} [\![v]\!]$.

It is easy to prove that rules (1)–(6) terminate on any input of the form $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$ with all terms linear, resulting in a combination of unions and intersections of finite sets of patterns which, by systematic application of distributivity of $\cap$ over $\cup$ and computation of symbolic $\cap$ operations, results in a finite set of $\Sigma \cup \Sigma^\#$-patterns denoting the same language as $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$.

Case (2), i.e., a problem $\{t\} - \{t\sigma_1, \ldots, t\sigma_n\}$ where some terms are non-linear but the σ_i are linear can be reduced to an intersection of cases of the form $\{t\} - \{t\sigma\}$ where $\sigma = \{\overline{x{:}s} = \overline{v}\}$ is linear. Then we can apply the single rewrite rule $\{t\} - \{t\sigma\} \to \{t\{x_i{:}s_i \mapsto w\} \mid x_i{:}s_i \in \overline{x{:}s},\ w \in \{x_i{:}s_i\} - \{v_i\}\}$, which, since σ is linear, reduces the problem to computing the differences $\{x_i{:}s_i\} - \{v_i\}$ between linear terms. The proof of correctness of this rule is entirely analogous to that for rule (3) of the linear case above and is left to the reader.

The correctness of the transformation in case (3) reduces to the observation that if $\sigma(x)$ is non-linear but the sort s of one the variables y occurring more

than once in $\sigma_i(x)$ is finite with $T_{\Sigma\cup\Sigma^\#,s} = \{u_1, \ldots, u_k\}$, then $[\![t\sigma]\!] = [\![t\sigma\{y \mapsto u_1\}]\!] \cup \ldots \cup [\![t\sigma\{y \mapsto u_k\}]\!]$.

The correctness of the Lassez-Marriott algorithm is proved in detail in [13] for the unsorted case and has a straightforward extension to the order-sorted case when the signature is $\Sigma \cup \Sigma^\#$. For references on the complexity analysis of this algorithm and a discussion of why it is considerably more efficient than similar algorithms in [19,18,7], we refer the reader to the detailed discussion by Pichler [18], where a more general —but computationally more costly— algorithm is given which can successfully compute pattern solutions whenever they exist for disjunctions of difference problems of the form $\{t_1\} - \{t_1\sigma_1^1, \ldots, t_1\sigma_{n_1}^1\} \cup \ldots \cup \{t_k\} - \{t_k\sigma_k^1, \ldots, t_k\sigma_{n_k}^k\}$. This is a key part of more general, but also more costly, algorithms that, given any Boolean expression *Bexp* involving term patterns, can *decide* whether $[\![Bexp]\!]$ is a pattern language and in the affirmative case can compute its pattern representation [19,18,7]. Instead, our algorithm —which generalizes that of Lassez and Marriott to order-sorted patterns— can only decide if $[\![t]\!] - [\![t_1, \ldots, t_n]\!]$ is an order-sorted pattern language and in the affirmative case can construct its explicit representation (see Thm. 4.1 in [13]). Since in practice many application problems can be expressed in the form $\{t\} - \{t_1, \ldots, t_n\}$, we consciously trade off the extra generality of the algorithms in [19,18,7] for the considerably greater efficiency of the Lassez-Marriott one [13].

A last practical issue is reducing the size of solutions of Boolean operations on finite sets of order-sorted patterns. That is, the resulting solution $\{t_1, \ldots, t_n\}$ may be bigger than necessary. This problem can be addressed by the application of two size-reducing and language-preserving rewrite rules, namely:

1. $\{t_1, \ldots, t_n\} \to \{t_2, \ldots, t_n\}$ *if* $t_2 \succcurlyeq t_1$, where we use associativity commutativity of set union to make the order of t_1, t_2 in the set immaterial, and where, by definition,[2] $t \succcurlyeq t'$ iff $t' = t\sigma$ for some substitution σ.
2. $\{t_1, \ldots, t_n\} \to \{u, t_3, \ldots, t_n\}$ *if* $\{u\} = lgg_{\Sigma\cup\Sigma^\#}(t_1, t_2) \wedge \{u\} - \{t_1, t_2\} = \emptyset$, where we require that the order-sorted *least general generalization* of t_1, t_2, denoted $lgg_{\Sigma\cup\Sigma^\#}(t_1, t_2)$ [1], which could be a set of terms, is actually a singleton set, and is furthermore "tight," i.e., $[\![u]\!] = [\![t_1, t_2]\!]$.

For example, the result of the difference $\{x{:}B\} - \{b\} = \{f(z{:}B^\#), f(z'{:}A), a\}$ for the signature $\Sigma \cup \Sigma^\#$ in the Introduction is reduced to $\{f(y{:}B), a\}$ by rule (2).

C Order-Sorted Difference Algorithm Evaluation

In this section we compare our order-sorted algorithm to solve a symbolic difference problem to the many-sorted algorithm into which —as explained in Section 5— such an order-sorted difference problem can be transformed. In general, a complete experimental comparison is impossible, since there are infinitely many signatures, each typically generating an infinite set of term patterns up to renaming. Thus, our goal with these experiments is not any kind of "proof," but

[2] We use the order in the same direction as in, e.g., [13], so that $\succcurlyeq$ is the "more or equally general than" relation. Other authors use $\preccurlyeq$ for the *same* relation.

rather to provide evidence that in non-toy examples our order-sorted algorithm is both more expressive and more performant.

```
fmod NAT-SIG is
  sorts Nat NzNat Zero .
  subsorts Zero NzNat < Nat .
  op 0 : -> Zero .
  op s_ : Nat -> NzNat .
  op _+_ : Nat Nat -> Nat .
  op _+_ : NzNat Nat -> NzNat .
  op _+_ : Nat NzNat -> NzNat .
  op _*_ : Nat Nat -> Nat .
  op _*_ : NzNat NzNat -> NzNat .
endfm
```

```
fmod INT-SIG is
  protecting NAT-SIG .
  sorts Int NzInt .
  subsort Nat < Int .
  subsorts NzNat < NzInt < Int .
  op -_ : Int -> Int .
  op -_ : NzInt -> NzInt .
  op _+_ : Int Int -> Int .
  op _*_ : Int Int -> Int .
  op _*_ : NzInt NzInt -> NzInt .
endfm
```

```
fmod RAT-SIG is
  protecting INT-SIG .
  sorts Rat NzRat .
  subsort Int < Rat .
  subsorts NzInt < NzRat < Rat .
  op _/_ : Rat NzRat -> Rat .
  op _/_ : NzRat NzRat -> NzRat .
  op -_ : Rat -> Rat .
  op -_ : NzRat -> NzRat .
  op _+_ : Rat Rat -> Rat .
  op _*_ : Rat Rat -> Rat .
  op _*_ : NzRat NzRat -> NzRat .
endfm
```

```
fmod COMPLEX-RAT is
  protecting RAT-SIG .
  sorts Cpx Imag NzImag NzCpx .
  subsort Rat < Cpx .
  subsort NzRat NzImag < NzCpx .
  subsorts NzCpx < Imag < Cpx .
  subsorts Zero < Imag .
  op _i : Rat -> Imag .
  op _i : NzRat -> NzImag .
  op -_ : Cpx -> Cpx .
  op -_ : NzCpx -> NzCpx .
  op _+_ : Cpx Cpx -> Cpx .
  op _+_ : NzRat NzImag -> NzCpx .
  op _*_ : Cpx Cpx -> Cpx .
  op _*_ : NzCpx NzCpx -> NzCpx .
  op _/_ : Cpx NzCpx -> Cpx .
  op _# : Cpx -> Cpx .
  op |_|^2 : Cpx -> Rat .
  op |_|^2 : NzCpx -> NzRat .
endfm
```

`COMPLEX-RAT` Signature

For greater generality in comparing these two algorithms, we might have randomly generated signatures from the space of all order-sorted signatures and further randomly generated terms in each signature. While this might seem more convincing, the majority of random signatures would never be used for any practical purpose. Thus, in our experiments we have favored practicality of examples over generality. To ground the discussion, we fixed a signature `COMPLEX-RAT`

which is part of an algebraic specification of the complex rational numbers. This signature —adapted from [10]— or very similar ones have been used in actual programming and verification tasks. Furthermore, it displays a substantial degree of subsorting, which will let us usefully compare the two algorithms (since they are essentially the same in the many-sorted case).

After fixing a signature, there is still the matter of generating terms to be inputs to the difference algorithms. In `COMPLEX-RAT`, we generated a set T of random terms via the following process: Given a desired term depth $n >= 0$, let a counter i be set to 0. While $i < n$, randomly select a non-constant operator for each open node in the term tree (the empty tree is open by default) and increment i. For each open node where $i = n$, randomly select a variable with 70% probability or a constant operator with 30% probability. For the experiments in this appendix, we set $0 \leq n \leq 2$.

Given a set of terms T randomly generated as above, we computed all pairs $(t_1, t_2) \in T^2$ and further generated the pattern operations $[\![t_1]\!] - [\![t_2]\!]$. Finally, these pattern operations can be input into our order-sorted algorithm as well as their many-sorted transformed versions. The metric used for comparison is the number of rewrites as counted by the Maude rewrite engine used to implement these two algorithms. This metric is useful because it is invariant across different computers and lets us abstract away from implementation differences.

The many-sorted transformed version of a problem was obtained by applying the sharpening transformation of Section 4 to the signature `COMPLEX-RAT` and to the pattern operation to be solved. Since many-sorted signatures are just a special case of order-sorted signatures, we can plug the transformed inputs into our order-sorted algorithm and it will compute results in a strictly many-sorted way. In general, since the transformation of an order-sorted difference problem into a many-sorted one described in Section 5 causes each order-sorted term to be expanded into a set of many-sorted ones, we can expect that the order-sorted difference algorithm will perform better, especially in cases with deep subsorting and multiple variables. Another side effect of this transformation is that answer sizes will likely be larger, since each order-sorted term may expand into several many-sorted ones.

In all, we computed roughly a thousand term difference problems using both our order-sorted algorithm and the many-sorted transformed problem and algorithm. On the whole, in the signature `COMPLEX-RAT`, with a thousand difference problems containing terms ranging from size zero to two, the many-sorted algorithm on average needed about 1,000 times as many rewrites as our order-sorted algorithm, with the median being 55 times as many rewrites.

Each dot in the figure below represents a randomly chosen difference problem among the roughly a thousand ones we generated. The dot's x-coordinate gives the number of rewrites needed to solve the problem using the order-sorted algorithm, while its y-coordinate gives the corresponding number of rewrites when the problem is reduced to a many-sorted one and the many-sorted algorithm is used. Note the difference in scale: the x-axis continues until 70,000 rewrites while the y-axis goes all the way to 800,000 rewrites.

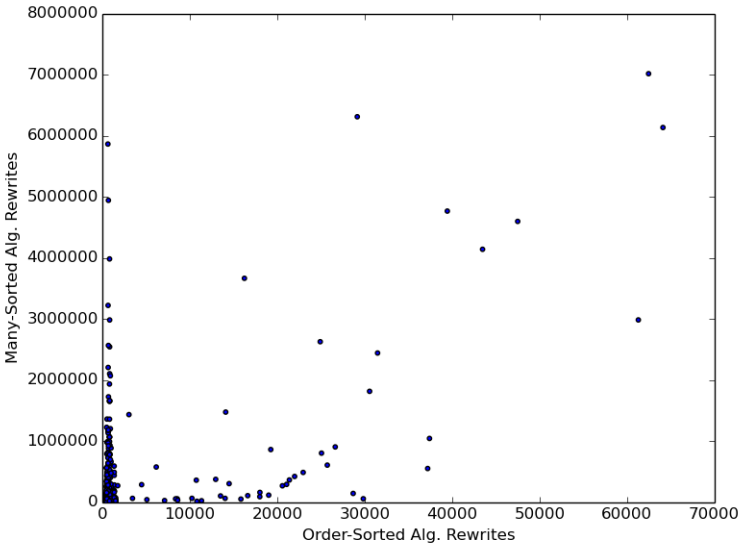

Fig. 1. Rewrites Used by Order-Sorted Vs. Many-Sorted Algorithm

OS Rewrites	MS Rewrites
23103	35811474
24937	49010016
53823	1233545932
69359	12619028
69765	14073907001
76389	21151931
78891	23954195214
86061	29805843801
91952	155444847
98377	5105671052
125465	12175065031
152156	152431926
192685	479356113
233652	713057516
340468	15062845092
664733	2618219

Fig. 2. Experimental Data Outliers

Actually, the full graph of the data would scale the axes even further, so for formatting purposes, some of points were removed and included in the table in Figure 2. If included in Figure 1, the difference in scale would become so great that the graph would become almost useless as a visual representation of the data. To better understand what is happening with these outlier points, we consider two examples: the outlier with the maximum x-value and the corresponding one with the maximum y-value.

The maximum x-value reported in our data set is 664733, with 2618219 the corresponding y-value and $2618219/664733 \approx 4$ the smallest y/x ratio in Figure 2. The operation in question is $\{\texttt{-C:Cpx}\} - \{\texttt{-(R:Rat*0)}\}$. Here, because of the deep sorting and unifiability of the terms, the order-sorted algorithm must consider many possible operators that `C` could instantiate into that do not unify with `R:Rat*0`. On the other hand, since there are only two variables, the many-sorted expansion does not generate as many patterns as when the problem contains many variables.

Alternatively, we may consider the maximum y-value: 29805843801, with 86061 its corresponding x-value and $29805843801/86061 \approx 346334$ the largest y/x ratio in Figure 2. In this case, the pattern operation is given by:

```
{R1:Rat*R2:Rat} −
        {((N1:Nat+N2:Nat)*(N3:Nat+N4:Nat))*((N5:Nat+N6:Nat)+s(0))}
```

As predicted above, this particular operation with eight variables expands into a huge number of cases—1,323 separate cases in fact. Furthermore, each case may require several levels of descent to verify how the terms overlap.

In summary, the data collected from our experiments show that for real signatures of interest even small input terms may create a huge difference in performance when choosing between the many-sorted and order-sorted algorithms.

D Case Study

We use the pattern operation algorithms presented in this paper to analyze the sufficient completeness of a specification of addition and multiplication in the natural numbers and integers. We first explain the notion of sufficient completeness and how term differences can settle sufficient completeness problems. We assume acquaintance with the following notions: (i) order-sorted rewrite rule $l \to r$, and rewrite relation $\to_R$ associated to a set R of such rules; (ii) termination of the $\to_R$ relation. For more details about these notions see, e.g., [2].

The sufficient completeness problem arose as the question of whether in an equational program (Σ, E) defining several recursive functions each such function has been *fully defined* by the equations E, where for execution purposes each equation $t = t'$ in E is oriented from left to right as a rewrite rule $t \to t'$. More generally, this problem can be posed for a *terminating* rewrite-rule-based program (Σ, R) as follows: we split the order-sorted signature Σ as a disjoint union $\Omega \uplus \Delta$, where Δ are the so-called *defined symbols*, and Ω the *constructor symbols*. To simplify the exposition we assume that each subsort-polymorphic

family of function symbols $f_{[s]}^{[s_1]\dots[s_n]}$ in Σ is fully included in either Ω or Δ. Let $Irr_R \subset T_\Sigma$ denote the subset of those ground Σ-terms that are *irreducible* by the rules R, i.e., $t \in Irr_R$ iff $(\nexists)\ u\ \ s.t.\ \ t \to_R u$. The we call (Σ, R) *sufficiently complete* with respect to Ω iff $Irr_R \subseteq T_\Omega$. Since (Σ, R) is assumed terminating, this means that each ground Σ-term t always evaluates to a constructor term, capturing the idea that the symbols in Δ have been fully defined by the rules R.

Sufficient completeness can then be boiled down to the following equivalent property, expressed in the lemma below, whose easy proof is left to the reader:

Lemma 4. *Let (Σ, R) be a terminating rewrite rule program with $\Sigma = \Omega \uplus \Delta$ a decomposition into constructors and defined symbols. Then (Σ, R) is sufficiently complete with respect to Ω iff for each $f : s_1 \dots s_n \to s$ in Δ and each $u_i \in T_{\Omega,s_i} \cap Irr_R$, $1 \leq i \leq n$, $f(u_1 \dots u_n) \notin Irr_R$.*

To cast the sufficient completeness problem as a symbolic term difference problem we use Lemma 4 above and define a simple signature transformation $\Sigma \mapsto \Sigma_\Delta$ as follows: the poset of sorts of Σ_Δ extends the poset $(S, \leq)$ of Σ by adding to each connected component $[s]$ of Σ a new sort $d_{[s]}$ with $d_{[s]} > \top_{[s]}$. The operators of Σ_Δ are those of Ω plus for each $f_{[s]}^{[s_1]\dots[s_n]} \subseteq \Delta$ an operator $f : d_{[s_1]} \dots d_{[s_n]} \to d_{[s]}$. Note that for each $s \in S$ we then have $T_{\Sigma_\Delta,s} = T_{\Omega,s}$. What this achieves, is that for any $f : s_1 \dots s_n \to s$ in Δ the ground instances of the Σ_Δ-term $f(x_1{:}s_1, \dots, x_n{:}s_n)$ always instantiate the variables $x_1{:}s_1, \dots, x_n{:}s_n$ by constructor terms. To simplify the exposition let us assume —this assumption can be omitted but a somewhat more complex formulation is then needed— that for each defined symbol f there is an $f : s_1 \dots s_n \to s$ with $s_1 \dots s_n s$ biggest possible among the typings of operators in $f_{[s]}^{[s_1]\dots[s_n]} \subseteq \Delta$. Define $R_f = \{l \to r \in R \mid l = f(u_1 \dots u_n)\ \ s.t.\ \ u_i \in T_\Omega(X),\ 1 \leq i \leq n\}$.

Then, it is immediate that if we can show that the symbolic difference of Σ_Δ-patterns $\{f(x_1{:}s_1, \dots, x_n{:}s_n)\} - \{l \mid l \to r \in R_f\}$ equals $\emptyset$, then for each $u_i \in T_{\Omega,s_i} \cap Irr_R$, $1 \leq i \leq n$, $f(u_1 \dots u_n) \notin Irr_R$ and therefore, by Lemma 4, assuming R terminating, (Σ, R) is sufficiently complete with respect to Ω. Note that if $Irr_R = T_\Omega$, the emptiness of this symbolic difference is a necessary and sufficient condition for the sufficient completeness of a terminating (Σ, R).

Let us see some examples. Consider the following Maude specification of the natural numbers:

```
fmod NATS is
  sorts Nat NzNat Zero .
  subsorts Zero NzNat < Nat .
  op 0 : -> Zero [ctor] .
  op s_ : Nat -> NzNat [ctor] .
  op _+_ : Nat Nat -> Nat .
  op _*_ : Nat Nat -> Nat .
  vars N M : Nat .
  eq N + 0 = N .
  eq (s N) + (s M) = s s (N + M) .
```

```
  eq N * 0 = 0 .
  eq (s N) * (s M) = s (N + (M + (N * M))) .
endfm
```

In a Maude specification each operator is preceded by the keyword `op`, and each operator which is a constructor is declared with the `ctor` attribute. In all, there are two defined operators and two constructors.

To achieve the $\Sigma \mapsto \Sigma_\Delta$ transformation, in the above signature `NATS`, we would add the sort and subsort declarations:

```
  sort dNat .
  subsort Nat < dNat .
```

and change the declarations for defined operators `_+_` and `_*_` to:

```
  op _+_ : dNat dNat -> dNat .
  op _*_ : dNat dNat -> dNat .
```

To check that `_+_` is sufficiently complete we perform the query:

$$\{\texttt{N + M}\} - (\{\texttt{N + 0}\} \cup \{\text{s N} + \text{s M}\})$$

in the transformed signature Σ_Δ (of course itself internally transformed into $\Sigma_\Delta \cup \Sigma_\Delta^{\#}$), which yields the solution set:

$$\{\texttt{0 + s K}\}$$

This pattern represents an infinite set of constructor instances of the `_+_` operator for which `_+_` is not defined. Indeed, if we check the function definition:

```
  eq N + 0 = N .
  eq (s N) + (s M) = s s (N + M) .
```

we have covered the cases where the second input is a zero and where both inputs are non-zero, but not the case where the first input is zero and second non-zero! Using the above information, one additional equation solves the issue:

```
  eq 0 + s N = s N .
```

We can now compute in the transformed signature the symbolic difference: $\{\texttt{N + M}\} - (\{\texttt{N + 0}\} \cup \{\texttt{s N + s M}\} \cup \{\texttt{0 + s N}\})$ which is indeed empty.

In the same way, we can check the sufficient completeness of `_*_` by performing in Σ_Δ the symbolic difference:

$$\{\texttt{N * M}\} - (\{\texttt{N * 0}\} \cup \{\texttt{s N * s M}\})$$

which yields the result set:

$$\{\texttt{0 * s K}\}$$

Going back to our multiplication definition:

```
  eq N * 0 = 0 .
  eq (s N) * (s M) = s (N + (M + (N * M))) .
```

we see that we have made the same mistake we did before when defining addition. Again, a single new equation completes the definition:

```
  eq 0 * s N = 0 .
```

Happily, the symbolic difference below is also now empty:

$$\{\texttt{N * M}\} - (\{\texttt{N * 0}\} \cup \{\texttt{s N * s M}\} \cup \{\texttt{0 * s N}\})$$

Let `NATS-FIXED` be the modified module which extends `NATS` by adding these additional equations. The above check means that we have now reduced the sufficient completeness of `NATS-FIXED` to proving that its equations, oriented as rewrite rules, are terminating.

As an additional example, let us check the sufficient completeness of the following Maude specification of integer addition and multiplication:

```
fmod INTS is
  protecting NATS-FIXED .
  sorts Int NzNeg .
  subsort Nat NzNeg < Int .
  op -_ : NzNeg -> NzInt [ctor] .
  op _+_ : Int Int -> Int .
  op _*_ : Int Int -> Int .
  vars I J : Int .
  vars N M : Nat .
  vars N' M' : NzNat .
  eq 0 + I = I .
  eq s N  + - s 0  = N .
  eq s N  + - s M' = N + - M' .
  eq - N' + - M' = - (N' + M') .
  eq 0 * I = 0 .
  eq - N' * - M' = N' * M'  .
  eq - N' *   M' = - (N' * M') .
  eq M' * - N' = - (M' * N') .
endfm
```

Note the import declaration on the second line: `protecting NATS-FIXED`. Here we are importing all the definitions and declarations from our newly sufficiently complete natural number specification. A modular approach to specification building not only saves time but allows us to verify each part of our specification incrementally.

As before, we must first transform our signature by adding an additional sort `dInt`, an additional subsort relation: `Int < dInt`, and giving sort `dInt` to the arguments and result of the two defined operators `_+_` and `_*_`.

For integer addition we now generate the pattern:

$$\{\texttt{I + J}\} - (\{\texttt{0 + I}\} \cup \{\texttt{s N + - 0}\} \cup \{\texttt{s N + - M'}\} \cup \{\texttt{- N + - M}\} \cup \{\texttt{N + 0}\} \cup \{\texttt{s N + s M}\} \cup \{\texttt{0 + s N}\})$$

yielding the result:

$$\{\texttt{- s N + M}\}$$

A cursory examination of the definition of integer addition reveals a mistake:

```
eq 0 + I = I .
eq s N  + - s 0  = N .
eq s N  + - s M' = N + - M' .
eq - N' + - M' = - (N' + M') .
```

Here we see that a case for a negative integer in the first position is missing. However, a simple new equation suffices to complete the definition:

```
eq - N' + M = M + - N' .
```

We can then check that the resulting pattern difference expression associated to the new definition is indeed empty.

Checking the sufficient completeness of integer multiplication is also straightforward. We just check that the symbolic difference:

$$\{\texttt{I * J}\} - (\{\texttt{0 * I}\} \cup \{\texttt{- N' * M'}\} \cup \{\texttt{N' * - M'}\} \cup \{\texttt{- N' * - M'}\} \cup \{\texttt{N * 0}\} \cup \{\texttt{s N * s M}\} \cup \{\texttt{0 * s N}\})$$

is empty. However, in this case, the algorithm returns the pattern:

$$\{\texttt{- N' * 0}\}$$

From the definition of multiplication we can see that no equation applies if the first term is negative and the second is zero. Thus, we need to add the equation:

```
eq - N' *   0    = 0 .
```

Finally, we can check that the solution for the new definition of multiplication is empty. Thus, the check of sufficient completeness for this corrected definition of integer operations reduces to proving the termination of its rewrite rules.